TOWARDS A COMPREHENSIVE FRAMEWORK FOR CO-SIMULATION OF DYNAMIC MODELS WITH AN EMPHASIS ON TIME STEPPING

A Thesis
Presented to
The Academic Faculty

by

Matthias Hoepfer

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
August 2011

TOWARDS A COMPREHENSIVE FRAMEWORK FOR CO-SIMULATION OF DYNAMIC MODELS WITH AN EMPHASIS ON TIME STEPPING

Approved by:

Dr. Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Neil Weston
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Brian German
School of Aerospace Engineering
Georgia Institute of Technology

Mr. Frank Ferrese
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Vitali Volovoi
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: August 2011

ACKNOWLEDGEMENTS

The work presented in this dissertation would not have been possible had it not been for the invaluable help and support from many people at the GT Aerospace System Design Lab (ASDL). I would like to express my sincere gratitude to all of them.

First and foremost, I would like to thank my adviser and committee chair Dr. Dimitri Mavris for his continuous encouragement, support, guidance, and inspiration over the years. His broad knowledge, unique insight on systems design, enthusiasm on new ideas and his great personality make him a very capable adviser and I cannot express how much I have benefitted from his leadership.

I would also like to show my great gratitude and appreciation to Dr. Neil Weston, Dr. Brian German, and Dr. Vitali Volovoi for their kindness, patience, invaluable suggestions, and guidance as my committee members.

I am particularly grateful to the Office of Naval Research (ONR) for funding and supporting this work and to Mr. Frank Ferrese and Mr. Anthony Seman for their insightful opinions and feedback.

I also want to thank fellow ASDL student Burak Bagdatli for deriving the equations of motion of the triple pendulum.

Last but most definitely not least, I also want to extend my very special appreciation to Mr. Bassem Nairouz, fellow Ph.D. student at the ASDL. He was the person who made it all possible: The double pendulum idea, the modeling and testing, the splitting and recombining, and all his feedback about, and insights into, the issues of modeling and simulation were of invaluable help, and the true enablers for this work.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii
LIST OF FIGURES v
LIST OF SYMBOLS AND ABBREVIATIONS viii
Summary ix
CHAPTER 1 Introduction And Motivation 1
1.1 Introduction 1
1.2 Problem Identification 16
CHAPTER 2 Dynamic Models 20
2.1 Definitions and Basics of Dynamic Models 20
2.2 A Short Discourse into Discrete Event Simulation and QSS 26
2.3 Discrete Time Simulation of Continuous Time Dynamic Systems 29
2.3.1 Basic principles 29
2.3.2 Numerical Integration 32
2.3.2.1 Euler's method 33
2.3.2.2 Heun's method / Runge-Kutta 2 37
2.4 Adaptive time step 42
2.4.1 Historical Development Of Time Stepping Methods 45
2.4.2 Time-Stepping Algorithms And The Runge_Kutta-Fehlberg 2/3 (RKF23) method 58
CHAPTER 3 Co-Simulation Of Dynamic Models 69
3.1 Monolithic Models 69
3.2 Co-Simulation 75
3.3 Issues With Co-Simulation 81
3.3.1 Data exchange synchronization 85
3.3.2 Coupling Between Models 85
3.3.3 Constraints Between Models 87
3.3.4 Simulation Stability 89
3.3.5 Singularities 91
3.3.6 Stiff Systems 92
3.3.7 Hardware-In-The-Loop Systems 93
CHAPTER 4 Application Of Time Stepping Algorithm To Co-Simulation Of Dynamic Models 96
4.1 Time Stepping In Co-Simulation 100
4.2 Co-Simulation Setup and Method Application 108
4.2.1 The double pendulum 108
4.2.2 Double pendulum co-simulation setup testing 113
4.2.2.1 Step 1: Compare monolithic and split model 113
4.2.2.2 Step 2: Test for model time step stability 115
4.2.3 First attempt at time stepping with second derivative as parameter 116
4.2.4 Application of RKF23 as time stepping algorithm in "Black Box" simulation 121
4.2.4.1 Step 3: Programming and testing the RKF23 algorithm 121
4.2.4.2 Step 4: Apply RKF23 algorithm to split model, single bar as reference 121
4.2.4.3 Step 5: Test RKF23 algorithm for different dynamics 126
4.2.5 Extension of RKF23 application to co-simulation 132

4.2.5.1 Step 6: Apply algorithm to all state variables, using minimum of all time steps 132
4.2.6 Multi-rate Application of a Numerical Integration Algorithm to Co-Simulation 138
4.2.6.1 Step 7: Apply RKF23 to all variables, advance each variable independently 138
4.2.7 The triple pendulum 144
4.2.7.1 Behavior with increased state frequency 164
4.2.8 Additional use of RKF time stepping algorithm 166
4.2.8.1 Determination of a fixed time step 167
4.2.8.2 Handling of varying dynamics 168
4.3 Further Issues And Considerations 169
4.3.1 Trajectory interpolation 170
4.3.2 Determination of the slopes 176
4.3.3 Computational overhead 178
4.3.4 Access to sub-system internals („Grey Box“) 181
4.3.5 Hybrid simulation 183
4.3.6 Selection of RKF23 over other RKF methods 187
4.4 Conclusion 189
CHAPTER 5 Summary, Limitations, And Next Steps 192
5.1 Summary 192
5.2 Limitations 194
5.3 Next Steps 195
5.4 Conclusion 209
APPENDIX A RKF23 meta code 211
APPENDIX B RKF23 code for testing purpose 212
APPENDIX C Meta Code for individual time stepping approach 219
APPENDIX D equations of motion for the triple pendulum 220
APPENDIX E External Bezier algorithm in matlab 224
References 228

LIST OF FIGURES

Figure 1. Overview of yearly publications in the fields of simulation and modeling (retrieved from the Web of Science database, January 13, 2011) 5
Figure 2. Artist's impression of the new generation war ship on dispatch 9
Figure 3. Notional Overview of Integrated Power System (IPS) 10
Figure 4. Comparison between traditional and all-electric ship 11
Figure 5. Notional overview of system interactions of the DD(X) new generation warship 13
Figure 6. Sample trajectory of a real world continuous time system state 22
Figure 7. Sample discrete time state trajectory 24
Figure 8. Sample discrete event trajectory 27
Figure 9. Discretized continuous time state trajectory 30
Figure 10. Discretized state trajectory and approximation curve 31
Figure 11. Numerical integration approximation error 34
Figure 12. Improved slope derivation 39
Figure 13. "Slow" dynamics, large fixed time steps 44
Figure 14. "Fast" dynamics, small fixed time step 44
Figure 15. Varying dynamics: Large time steps in regions with "slow" dynamics, small time steps in regions with "fast" dynamics; an adaptive time step is advantageous in this situation, and required if error bounding and limitation is necessary 45
Figure 16. Notional graph of number of function calls vs. desired tolerance, RKF45 algorithm 60
Figure 17. Predictor step: Use $\Delta t/2$ slope to determine new point at $T + \Delta T$ 62
Figure 18. Corrector step: Use current, $\Delta T/2$, and ΔT slopes to determine new point at $T + \Delta T$ 63
Figure 19. New point and time step determination: Use error / discrepancy between predictor and corrector points to determine new time step; set new point at $T + \Delta T_{new}$ .. 63
Figure 20. Sample curve with "benign" curve settings (curve amplitude parameter = 2) 64
Figure 21. Sample curve with more "aggressive" curve settings (curve amplitude parameter = 1.05) 65
Figure 22. Sample curve with very "aggressive" curve settings (curve amplitude parameter = 1.0000001) 65
Figure 23. Comparison for numerical solution between fixed and adaptive time step 68
Figure 24. Notional depiction of a monolithic model (modeled in Simulink) 70
Figure 25. Notional block representation of monolithic model 71
Figure 26. Monolithic models executed in parallel, no interactions, no feedbacks 72
Figure 27. Transformation of a monolithic system into a co-simulation 76
Figure 28. Notional co-simulation setup 77
Figure 29. Notional graph of root mean square error (RMSE) as f(system eigenfrequency, time step). Note triple logarithmic scale 84
Figure 30. Causal conflict in refrigeration cycle model 87
Figure 31. Instability as a result of time steps chosen too large 91

Figure 32. General description of real world time, global time step, and local time step 102
Figure 33. Timing for sub-model execution in a co-simulation 102
Figure 34. Time step to be determined in co-simulation setup 105
Figure 35. Time stepping scheme for integrated model execution 106
Figure 36. Double pendulum principal components 109
Figure 37. Double pendulum model in Simulink 110
Figure 38. Double pendulum upper bar Simulink model 111
Figure 39. Double pendulum lower bar Simulink model 112
Figure 40. Double pendulum lower and upper bar integrated into co-simulation setup. 113
Figure 41. Comparison monolithic – split model, upper bar 114
Figure 42. Comparison monolithic – split model, lower bar 115
Figure 43. Test of integrated model for time stepping robustness 116
Figure 44. Notional explanation of impact of second derivative on time step 118
Figure 45. Applied time step variation with second derivative method 119
Figure 46. Applied RKF23 algorithm to double pendulum model, upper bar as reference 122
Figure 47. Applied RKF23 algorithm to double pendulum model, lower bar as reference 122
Figure 48. Fitted third order polynomial for slope determination 124
Figure 49. Comparison between calculated slope and slope acquired directly from model 126
Figure 50. Double pendulum model “slow” configuration, upper bar 127
Figure 51. Double pendulum model “fast” configuration, upper bar 128
Figure 52. Double pendulum model “slow”, toggled to “fast” at 1s < T < 2s, then “slow” again, upper bar 131
Figure 53. Notional depiction of the variables under consideration as time stepping metrics 134
Figure 54. Notional explanation of new time step determination approach 135
Figure 55. Lowest time step used for both bars, “slow” dynamics 136
Figure 56. Lowest time step used for both bars, “fast” dynamics 136
Figure 57. Lowest time step used for both bars, “toggled” dynamics 137
Figure 58. Notional explanation of time step interpolation approach 139
Figure 59. Detailed description of time stepping and updating approach 140
Figure 60. Time step interpolation scheduling 141
Figure 61. RKF23 time step applied to both bars separately 142
Figure 62. Notional depiction of a triple pendulum model 145
Figure 63. Initial base run of triple pendulum model 146
Figure 64. Triple pendulum run with varied mass m1 = 100 149
Figure 65. Triple pendulum run with varied bar lengths l1 = 0.1, l3 = 100 150
Figure 66. Graph for single mass variations 153
Figure 67. Graph for single length variations 154
Figure 68. Contour plot for variation of l1 and l2, l3 = 0.1 157
Figure 69. Contour plot for variation of l1 and l2, l3 = 1.0 158
Figure 70. Contour plot for variation of l1 and l2, l3 = 10 158
Figure 71. Modulations of bar 1 onto bars 2 and 3 161

Figure 72. Modulations of bar 1 onto bars 2 and 3, different initial angles 162
Figure 73. Notional 3-element coupled dynamic system, one sub-system connected to fixed reference system 163
Figure 74. Notional 3-element coupled dynamic system, no connection to fixed reference system 163
Figure 75. Triple pendulum run with one bar at very high frequency 165
Figure 76. Amplification of high frequency behavior 166
Figure 77. Triple pendulum run with changing sub-system dynamics 168
Figure 78. Smooth trajectory example 171
Figure 79. Trajectory curve example with sharp drop 172
Figure 80. Smooth trajectory with different interpolation algorithms 174
Figure 81. Trajectory curve with sharp drop, different interpolation algorithms 174
Figure 82. Trajectory curve with sharp drop, mathematical equation 175
Figure 83. Implementing state derivatives directly from model (not from interpolation and approximation) 182
Figure 84. Time line scenario for discrete event simulation in continuous time model setup 185
Figure 85. Different ways of discrete even model synchronization with continuous tie models 186
Figure 86. Comparison between RKF23 and RKF45 algorithms for numerical solution of differential equation 188
Figure 87. Proposed extrapolation algorithm for time stepping 201
Figure 88. Sample FFT result for upper pendulum bar 202
Figure 89. Method to estimate error due to adaptive time step 203

LIST OF SYMBOLS AND ABBREVIATIONS

ASDL ..... Aerospace Systems Design Laboratory

DAE ..... Differential Algebraic Equation

DE ..... Differential Equation

FFT ..... Fast Fourier Transform

HIL ..... Hardware-in-the-Loop

IED ..... Integrated Electric Drives

IEP ..... Integrated Engineering Plant

IPS ..... Integrated Power System

IRIS ..... Integrated Reconfigurable Intelligent Systems

LCC ..... Life Cycle Cost

M&S ..... Modeling and Simulation

ODE ..... Ordinary Differential Equation

ONR ..... Office of Naval Research

R&D ..... Research and Development

RK .....General Reference to Runge-Kutta numerical integration method

RK2 ..... Runge-Kutta 2 fixed step numerical integration method

RK4 ..... Runge-Kutta 4 fixed step numerical integration method

RKF23 ..... Runge-Kutta-Fehlberg $2^{nd}/3^{rd}$ order predictor-corrector method

RKF45 ..... Runge-Kutta-Fehlberg $4^{th}/5^{th}$ order predictor-corrector method

RMSE ..... Root Mean Square Error

SUMMARY

Over the last two decades, computer modeling and simulation have evolved as the tools of choice for the design and engineering of dynamic systems. With increased system complexities, modeling and simulation become essential enablers for the design of new systems. Some of the advantages that modeling and simulation-based system design allows for are the replacement of physical tests to ensure product performance, reliability and quality, the shortening of design cycles due to the reduced need for physical prototyping, the design for mission scenarios, the invoking of currently non-existing technologies, and the reduction of technological and financial risks.

Traditionally, dynamic systems are modeled in a monolithic way. Such monolithic models include all the data, relations and equations necessary to represent the underlying system. With increased complexity of these models, the monolithic model approach reaches certain limits regarding for example, model handling and maintenance. Furthermore, while the available computer power has been steadily increasing according to Moore's Law (a doubling in computational power every 10 years), the ever-increasing complexities of new models have negated the increased resources available. Lastly, modern systems and design processes are interdisciplinary, enforcing the necessity to make models more flexible to be able to incorporate different modeling and design approaches.

The solution to bypassing the shortcomings of monolithic models is co-simulation. In a very general sense, co-simulation addresses the issue of linking together different dynamic sub-models to a model which represents the overall, integrated

dynamic system. It is therefore an important enabler for the design of interdisciplinary, interconnected, highly complex dynamic systems. While a basic co-simulation setup can be very easy, complications can arise when sub-models display behaviors such as algebraic loops, singularities, or constraints.

This work frames the co-simulation approach to modeling and simulation. It lays out the general approach to dynamic system co-simulation, and gives a comprehensive overview of what co-simulation is and what it is not. It creates a taxonomy of the requirements and limits of co-simulation, and the issues arising with co-simulating sub-models. Possible solutions towards resolving the stated problems are investigated to a certain depth. A particular focus is given to the issue of time stepping. It will be shown that for dynamic models, the selection of the simulation time step is a crucial issue with respect to computational expense, simulation accuracy, and error control. The reasons for this are discussed in depth, and a time stepping algorithm for co-simulation with unknown dynamic sub-models is proposed. Motivations and suggestions for the further treatment of selected issues are presented.

CHAPTER 1 INTRODUCTION AND MOTIVATION

1.1 Introduction

Historically, the design, testing and manufacturing of a new product or system required extensive prototyping and hardware involvement. This approach has some severe drawbacks and disadvantages. First, a model must be designed and built. This involves large amounts of cost and time. The model itself may not be flawless, thus rendering any testing results invalid. Sometimes, accurate small-scale models are difficult or impossible. Examples here are models used for flutter and vibrations testing, or aerodynamic models where Reynolds- and Mach numbers must be taken into account when scaling. Another disadvantage of the model building approach is that a model may only be changed or modified with great difficulties, and usually with more cost and time delay. Also, such models could not be "adaptive" in a sense that they would "learn" from model runs and adapt their behavior. A model also can only represent the current state of technology. There would be no feasible way to test or integrate a possible new technology. One more point to be mentioned here is the fact that any simulation that would lead to the destruction of the original system would also lead to the destruction of the model. Imagine here the testing of a military system's resistance against ammunition hits. Clearly, any such test would likely result in major damage to a model, rendering the testing a tedious procedure. Further, if the impact of any system's behavior on humans were to be tested appropriately, such human involvement would put anyone at risk that would be unlucky enough to be chosen as the test subject. Lastly, modern systems and systems-of-systems become too large and complex to be tested by any feasible

“hardware” modeling technique. To bypass all these disadvantages of physical model testing, computer simulation is commonly employed.

With the advent of digital computers, with their increased computational powers and possible interconnectivity, modeling and simulation (M&S) of new products and processes became an option to the tedious, costly, slow and often dangerous development and testing of new physical systems. Instead of building such physical models, the systems could now be modeled and simulated in a computer. M&S has since become the development tool of choice for many application fields. Performance could be modeled and forecasted, and the models could easily be changed to examine the impact of design variations. In fact, with more capable computers, probabilistic design with millions of iterations and variations is nowadays a common design methodology for new products. These computer models could be used to ensure product performance metrics, product reliability, and quality. Also, cost, being an increasingly important factor, could now be scrutinized and planned for much more detailed and accurately. Modeling and simulation further greatly reduced the design cycle times. While modeling a complete system in a computer may still require quite large up-front time, cost, and efforts, the actual design investigations can be performed very quickly once the model is built. Such investigations can then include radical new approaches, for example by infusing technologies that as of the time of the test may not even exist yet. This is commonly referred to as design space exploration: Expose the model to all possible sorts of modifications, and see which one will turn out to be the best design. These modifications can now include mission scenarios that were previously out of reach for any testing. Battlefield simulations are very easily and quickly set up and the impact of assumed enemy weapons could be

simulated without even knowing the exact specifications of such weapons. In general, such computer modeling and simulation approaches could bypass any of the disadvantages of physical systems models described before. No humans would need to be put on the line for testing any more (even though this is not always true either. For example, an airplane prototype will still need to be flow for the first time by trained test pilots. But the risk for these pilots is dramatically reduced due to extensive testing before the first prototype is even built). Also, increasingly complex and interconnected systems could now be modeled and tested. Modeling and simulation can also be used for training. A typical application for this is a flight simulator, where beginner pilots can learn to fly airplanes without the risk of damaging anything. Also, M&S will greatly improve failure and safety analyses. The beauty of M&S lies among others also in the ease of obtaining and measuring data. The data from M&S runs is readily available in the computer, and can be post processed accordingly. There are no sensors that need to be calibrated, installed, monitored, read out, etc. in order to obtain the data.

The above mentioned possibilities of implementing currently non-existing technologies and unknown scenarios made it possible to leapfrog the design of new systems. Systems that could use such technologies, and deal with such scenarios, could be developed only after modeling them in a computer was possible. Computer modeling and simulation were the enablers for such futuristic design paradigms. High technology such as smart weapon systems could not be implemented with the traditional design methods, and must make use of the new computer technologies. In order to stay technologically competitive, the capabilities of modeling and simulation for the analysis of more complex systems will need to be extended even further in the future.

Modeling and simulation of systems in a computer is not however solely limited to the design and development of physical systems. Since adequate computational power has become abundant and cheap enough to make it an all-available commodity, more and more domains have started to make use of the new capabilities. This ranges from microscopic to interstellar phenomena. Material behavior can be simulated, and new materials be developed, that possess tailored properties for certain applications. Blood flow through the human body can be better understood by simulating it in a computer. Behavioral science and economics also rely heavily on modeling and simulation, which help to understand psychological phenomena in the behavior of humans and animals, for example the different human behaviors in masses vs. in solitude. For problems like these, practical trials would hardly be feasible and, given the "laboratorial" character of trials, the results might be reasonably doubted by reviewers. Weather forecasting could be improved greatly by using images from satellites to forecast storms and flooding. Computational models are also used for earth models, for example to investigate the impact of global warming on the weather. Of course, such investigations require simulation since neither the weather nor the Earth's ecosystems can be modified in a controlled manner. Simulation spans all across those domains, up to the simulation of such cosmic events as the collision of two galaxies.

Another application for modeling and simulation evolved with economics and finance. Markets and market participant behaviors could now be more accurately represented, which in turn helped forecasting of economies. This also has implications for engineering. Nowadays, every engineer also has to keep cost in mind at all times. Product development cycles must be shortened in order to stay competitive. This has implications

on the finance of projects. Modeling of economic impacts on a project helps to evaluate the cost side of product development before it is even started, and helps to ensure that cost will not run out of bounds during the process, and financing will be secure all along the product design cycle. More fields that rely heavily on M&S are for example critical infrastructures, emergency management, manufacturing, sustainable future systems, and biomedical.

An indication of the increased importance of modeling and simulation can be to look at the number of publications concerning these two fields. Figure 1 gives an overview of the amount of literature published in the fields of modeling and simulation.

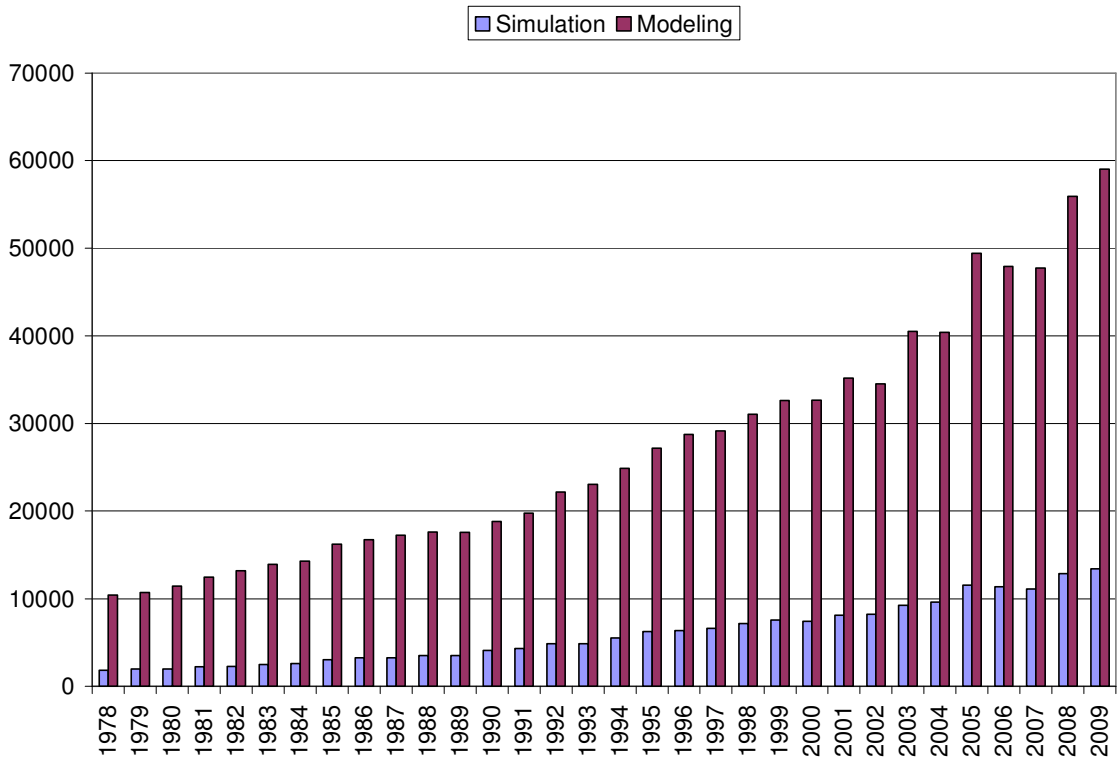

Figure 1. Overview of yearly publications in the fields of simulation and modeling (retrieved from the Web of Science database, January 13, 2011)

It can be clearly seen that over the last 30 years, the increased possibilities through the development of more and more powerful computers has sparked the interest in the subject. One reason is that, as mentioned before, no company can nowadays remain competitive without heavily investing in modeling and simulation. Also, cutting-edge research and development (R&D) is only possible through modeling and simulation. It can safely be said that the importance of modeling and simulation will likely increase in the future.

Modeling and simulation become more and more important for any company to develop and test their products, and at the same time the enabler for such progress, namely more and more powerful computers, have also been developed. However, since companies rely more and more heavily on modeling and simulation, and technologies and scenarios to be tested become more and more demanding, there is a tendency for the models and scenarios to become more and more complex. More and more features are implemented into the models, and more and more test cases need to be run to fully understand these complex models and their behavior. Therefore, despite the ever increasing power of modern computers (Moore's law predicts a doubling in the number of transistors integrated into a chip every two years, which translates almost linearly into increased processing speeds, memory capacities, etc.), the advantages of these faster computers are compensated by the also ever increasing complexity of the investigated models. The trend of increased model complexity is likely to continue in the future, as more and more functions and properties will need to be implemented into the models, and new and coming technologies will also need to be considered. This is reflected in Wirth's law, postulated by Swiss computer scientist Niklaus E. Wirth, which states that "Software

is getting slower more rapidly than hardware becomes faster". New software features and capabilities bog down even last-generation computers. In the abstract to his 1995 paper "A Plea for Lean Software" (Wirth 1995), Wirth writes "Memory requirements of today's workstations typically jump substantially--from several to many megabytes--whenever there's a new software release. When demand surpasses capacity, it is time to buy add-on memory. When the system has no more extensibility, it is time to buy a new, more powerful workstation. Do increased performance and functionality keep pace with the increased demand for resources? Mostly the answer is No. The author contends that software's girth has surpassed its functionality, largely because hardware advances make this possible. He maintains that the way to streamline software lies in disciplined methodologies and a return to the essentials. He explores the reasons behind software's increasing heft…". This however, must be seen in the relation to new user interfaces and unused and unneeded functionalities in software packages. In the context of simulation, it is natural that with increased computational power provided, the models inherently become more and more detailed and complex because it helps the engineer and designer to get more and more realistic images of the real world. The inherit danger is that such models start to bog down even the most powerful computers. It is therefore necessary to start considering an increase in simulation efficiency so as to be able to further use the increased calculation powers provided by each new generation of micro processors.

The increased complexity of computer models described above is demonstrated by systems like the IRIS (Integrated Reconfigurable Intelligent Systems) approach for the new generation destroyer of the US Navy and ONR, which was the initial project and problem that led to the investigations described in this thesis. The IRIS approach for the

new DD(X) naval destroyer ships can be seen as an archetypical example for the use and the requirement for computer modeling and simulation of a complex system. The Navy's DD(X) program was established to develop the next-generation, multi-mission destroyer for the US Navy (Gilmore 2005). This new generation destroyer is intended to allow for innovative and radical departures from traditional designs, and include much stronger focus on combat effectiveness, survivability, and resilience, while at the same time reduce design time and cost, and overall cost of ownership. The lower cost of ownership is mainly achieved through reduced manning, which in turn requires higher degrees of operations automation. This automation will have to work under the various operational and combat scenarios that the ship will encounter during its lifetime. This is therefore a field which is predestined to be investigated using computer modeling and simulation. Lower cost of ownership, but also increased effectiveness, survivability and resilience are also achieved through radical new technologies such as and integrated systems environment, increased stealth capabilities, advanced gun systems, wave-piercing hull shape, and others. These systems are currently under development in part at the Aerospace Systems Design Laboratory (ASDL) of the Georgia Institute of Technology Aerospace Department. Currently, two ships are being built concurrently by General Dynamics and Northrop Grumman, and scheduled to start service in 2012. An artist's impression of the new ship is shown in Figure 2.

Figure 2. Artist's impression of the new generation war ship on dispatch

In order to achieve its planned mission capabilities as described above, the design and development of the DD(X) class ships is extended by implementing two radical new design concepts. The first concept is that of an Integrated Power System (IPS), as described by Doerry et al. (Doerry 1996), see Figure 3. The IPS was initiated in 1994, and is often colloquially referred to as the "All Electric Ship". The underlying idea is to develop a ship whose propulsion and electrical systems are interrelated and coupled. This is in contrast to traditional ship designs, where two distinct and separated power systems are responsible for the ship's propulsion and electrical systems, respectively.

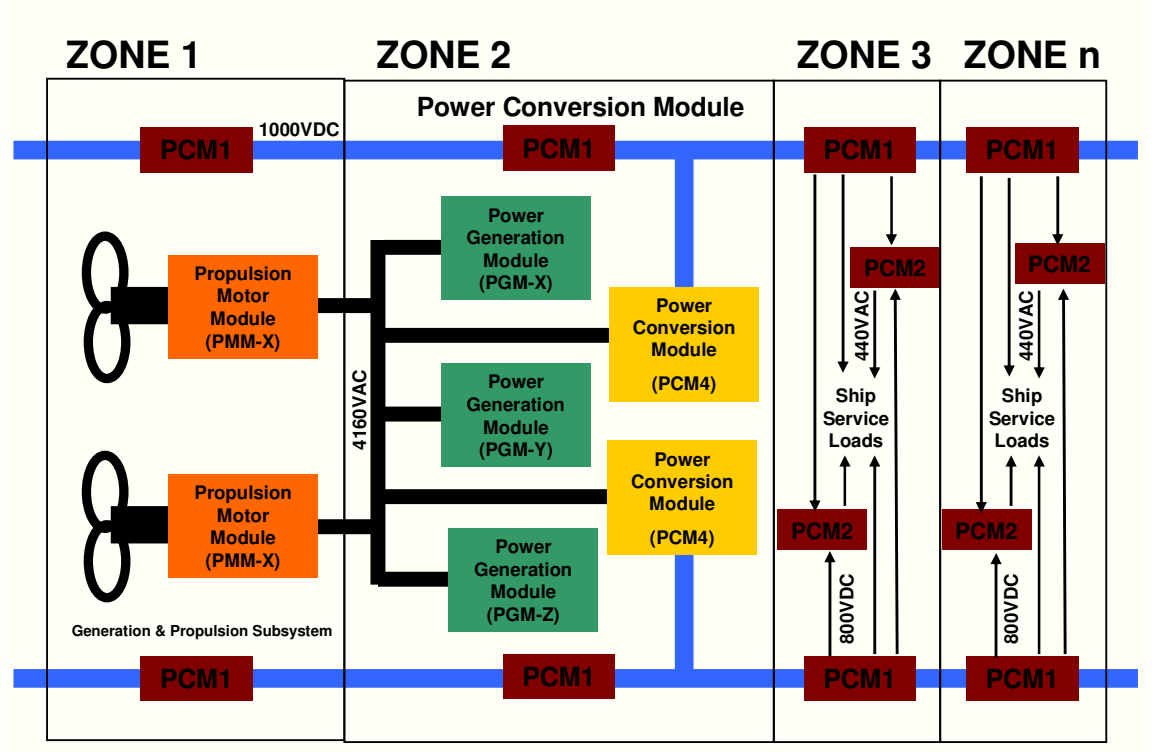

Figure 3. Notional Overview of Integrated Power System (IPS)

In the IPS approach, ship propulsion and power management are integrated into one system. The mechanical propulsion system is replaced by electric motors. Thus, the power generation units need only provide electricity, which is separated into propulsion and miscellaneous systems requirements. This power system is designed in a modular fashion, so that it can be designed and manufactured with interchangeable components. Figure 4 compares the traditional and new approach towards ship propulsion and electrical systems, and its immediate effect on the necessary manning requirements. The advantages of this approach, according to Doerry et al. (Doerry, 1996), are:

- Higher design and manufacturing efficiency and flexibility
- Greater operational efficiency
- Improved maintainability and flexibility for upgrades

- Improved Life Cycle Cost (LCC) and cost of ownership

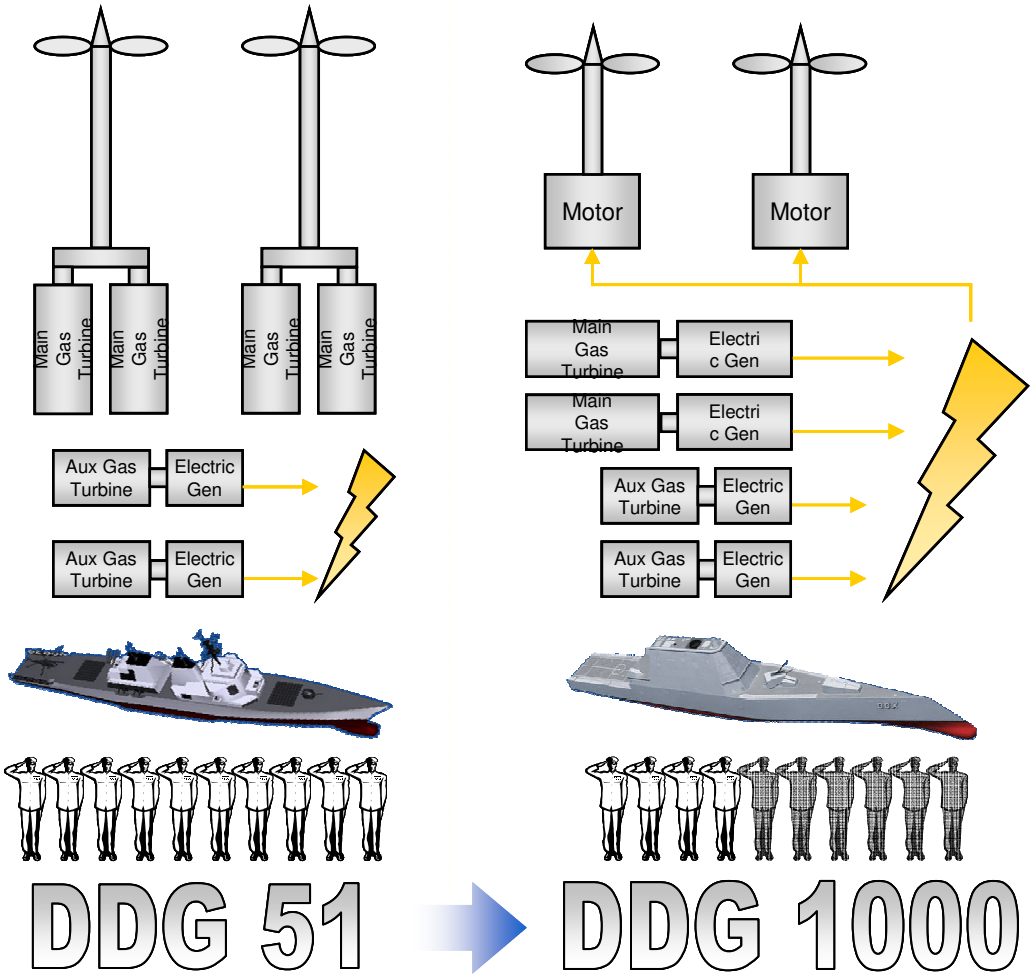

Figure 4. Comparison between traditional and all-electric ship

The second design concept is the Integrated Engineering Plant (IEP), as described in Dunnington et al. (2003), Lively et al. (2005), and Walks et al. (2005). With the IPS architecture as its underlying basic, the IEP implements a higher degree of monitoring, control and automation into the system by utilizing sensors and electric actuators throughout the entire ship. This is one main contributor to the reduced manning requirement for future combat ships by the US Navy. Besides the cost requirements, reduced manning also reduces the necessity for human involvement and decision-making,

thus reducing the errors that result from these involvements, and increase execution speeds and accuracies. Naturally, such an approach involves large automation and controls architectures.

The challenge that this project posed was to model such an approach as a computer simulation. The focus was on modeling and integrating three main components of the proposed approach. The three components work together in the following way:

- The electrical system is modeled to represent the electrical loads in the different zones of the IPS. In a real world scenario, these electrical loads will be employed dynamically as the ship fulfills its missions and goes through different mission scenarios, all of which will have an impact on how the various electrical loads will be used. When an electrical load is used, it will create waste heat which will have to be removed from the system. This is the task of the
- Fluid Network component. It is made up of heat exchangers which will absorb the heat created in the different heat loads of the electrical systems. The hot fluid will then be led through a network of piping which will feed the fluid into sea water chillers. These chillers convect the heat into the surrounding sea water, thus removing it from the system. The fluid is led through the piping network by redundant pumps, and a network of valves which will ensure proper resource allocation to each of the heat load exchangers. The valves are controlled by a
- Hierarchical Controller. This controller must ensure that all heat loads have sufficient supply of cooling water under varying mission scenarios and subsequent ship performance requirements. The challenge is that this resource allocation must also function in the case of fluid network damage, for example

through an external weapon hit. Under such circumstances, parts of the fluid piping network will be disabled and ruptured. The controller must make sure that such ruptures are isolated (to prevent the leakage of cooling fluid) and that nevertheless the remaining accessible heat loads can be supplied with sufficient amounts of cooling fluid. This is a highly dynamic problem, and the system constantly changes during operations.

The IRIS concept is modeled in a simulation environment that integrates the sub-systems into an overall simulation model, with which the control architecture can be developed and tested. Figure 5 shows a notional overview of the systems in the ship, and their interactions during the simulation. The integrated simulation also contains an HMI (Human Machine Interface) module which enables data readout and storage, and human interaction with the simulation during run time (e.g. to be able to change simulation parameters during the simulation run).

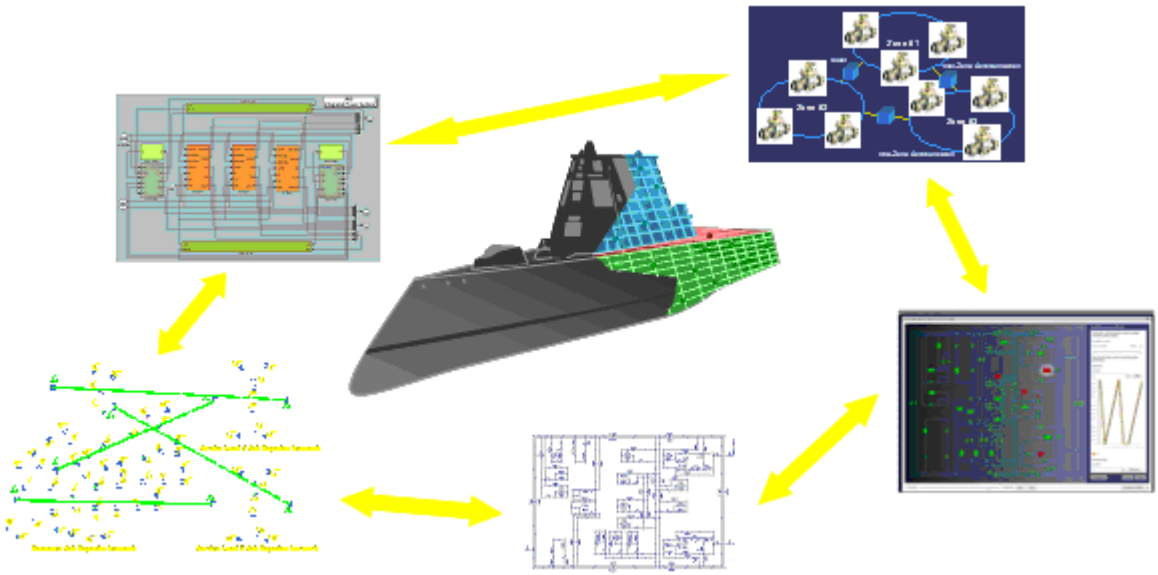

Figure 5. Notional overview of system interactions of the DD(X) new generation warship

It was mentioned earlier, that modeling and simulation is applicable in a vast array of applications. In the particular case of the IRIS simulation, the different sub-systems used in this setup are dynamic systems. This type of system will be assumed for the remainder of this thesis text. The use, modeling and simulation of such dynamic systems pose certain challenges that will need to be taken into account when setting up and executing the sub-system models in a digital computer. Since the underlying system equations can oftentimes not be solved in a closed form solution, an approximation to the function must be made. This approximation is a numerical integration of the underlying system equations. It will be shown that executing dynamic system models in a digital computer simulation requires a time stepping scheme that plays a critical role in the accuracy of the approximated results. The reasons for this will be laid out, and the implications for the integration of the sub-models will be discussed. The issue of time stepping for dynamic models is very widely discussed in the literature, and various algorithms exist that can effectively handle adaptive time stepping schemes for the approximation of the solutions.

The dynamic sub-systems introduced above are modeled as discrete "monolithic" models, meaning that they were modeled independently of each other each with their own modeling language and tool, and that they by themselves do not interact with each other. They only fulfill the functionality that is within their scope, and do not exchange any data among each other. The mathematical "internals" of these models is unknown; they are "Black Boxes" with respect to the knowledge about their internal mathematic structure. Hence classical approaches towards their solution during simulation execution are not

applicable. To get an overall model of the entire ship including all the desired systems, the sub-models must be integrated into an overall integrated simulation. This is commonly referred to as co-simulation (or parallel simulation, multi-rate or multi-time (when different time scales are considered), and multi-scale (when different dimensional scales are considered) simulation). This overall integrated simulation model will then itself be a dynamic system, with all the implications made for single dynamic system models.

When simulating dynamic systems (be it monolithic or combined into a co-simulation) in a digital computer, the real world continuous time can not be simulated directly. Rather, the time must be discretized into time steps at which the simulation model is evaluated. This time discretization (or *time stepping*) is of critical importance to the simulation results. The time step must be properly selected. The larger the time step is selected, the fewer computations must be executed, and computation time is saved. However, this has detrimental effects on the simulation accuracy and stability. However, making the time step too small will result in increased computational expense and, if the step is chosen too small, added error due to the limited resolution of numbers within a computer. In order to bypass this problem, adaptive time stepping algorithms have been developed that enable the error of a simulation to be controlled through varying the time step as needed during the simulation run. Such algorithms are well-developed for monolithic models whose system equations are known. Co-simulation however, adds a new layer of time stepping schemes due to the fact that not only do the sub-systems themselves have to be time stepped correctly, but also the exchange of data between the now linked sub-systems needs to be timed and time stepped correctly in order for the

simulation to deliver accurate results, and not to become unstable during the simulation execution. The time step that defines the exchange of data between sub-models is called the "global" time step. This thesis proposes an approach to setting this global time step in "Black Box" co-simulation environments by employing a method that has been proven to work well in numerical integration applications.

1.2 Problem Identification

In this text, two gaps are identified and addressed. The first one is the fact that co-simulation is well defined if the equations of the sub-models are given, and the integration can be performed numerically. This has been treated extensively in the literature. However, if this is not the case, and sub-model behaviors can only estimated through state data observations, the approach to co-simulation must be modified. To date, the literature does not deal with this type of problem in an in-depth manner. Hence, this text tries to frame co-simulation of dynamic sub-models with respect to the issues that arise from the integration of these models to an overall integrated simulation. The issues are partially derived from monolithic models, but the scope is extended through problems arising due to the inherent setup of co-simulation.

The second gap is more specific and the main focus of this text. Namely, it is that of how to set the global simulation time step within an integrated co-simulation of sub-models whose internal equations are unknown. As mentioned before, if the equations are known, then the problem can be treated with proven numerical algorithms. The question is how time steps in co-simulation, with their crucial importance to accuracy, stability, and other issues with dynamic models, can be set in a defined, quantitative manner when the proven numerical algorithms can not be applied in a straightforward manner due to the "Black Box" constraint, while taking into consideration the intrinsic problems that

come along with dynamic system co-simulation. This problem has not been tackled in the literature for general settings in which system equations of underlying systems, and/or the overall equations of the simulation, are unknown. Hence, this thesis tries to bridge this gap by introducing an algorithm that can help to determine time step settings under such conditions, taking into consideration the specific problems and issues that arise when monolithic dynamic models are integrated into an over co-simulation model, as described before. It will do so by applying a method and an algorithm that allows for adaptive time stepping from the field of numerical integration of differential equations to solve the problem of time stepping for co-simulation of "Black Box" dynamic systems in the field of co-simulation. It integrates knowledge of one disciplinary area into another. It must be understood, though, that this thesis can only be a "starter" text to introduce the application of an algorithm to the co-simulation time stepping problem. The discussion will be based on a sample "toy problem" which is used to develop and verify the approach. This means that the proposed algorithm will work for a small set of sample problem but it does not mean that it is universally applicable to all problems related of dynamic system co-simulation. To achieve this, future work will be proposed in the final chapter which, if realized, would lead to a more universal application of such an algorithm.

This thesis attempts to resolve the above mentioned issues, and fill the described gaps. The insights gained in this introductory chapter will provide the basis for the following discussions. The text is built up as follows:

Chapter 2 gives the basics about dynamic systems, and the issues that accompany them. It will describe the way monolithic dynamic models are usually set up, and the rules that describe their behavior. It will point out some of the main issues with dynamic models and their simulation on digital computers. It will also give a somewhat detailed introduction into the time stepping that is necessary for dynamic system simulation on digital computers. This will be the basis for the explanations and derivation of the method

proposed in Chapter 4. It must be understood though, that this chapter can only be an introductory one; the issues with dynamic system M&S are extremely widespread, and have been extensively scrutinized in the literature. Hence, a great part of the explanations will be short, and for further information, the respective literature will be referenced.

Chapter 3 extends the idea of monolithic dynamic system model to the co-simulation concept. Based on the findings in Chapter 2, it will transfer the issues of monolithic dynamic models, and discuss the differences and intricacies of co-simulation of dynamic systems. Co-simulation of dynamic systems has been extensively treated in the literature as well, but most such literature focuses on the situation where the underlying equations for the sub-models and the coupling conditions and equations are known. As mentioned before, such setups make the treatment of co-simulation much easier and more mathematically stringent. However, if the equations, and hence the dynamics, of the sub-models are not known, then the issue of co-simulation becomes a more complicated one. Chapter 3 will briefly describe the current state of co-simulation with known equations, and then go into more detail about the issue of unknown sub-model equations and what this means for ht simulation of an integrated model. This will also lay the groundwork for Chapter 4.

Chapter 4 will use the descriptions in the previous chapters to build and propose and algorithm that is derived from basic monolithic model simulation to try and adapt such an algorithm to the problem of co-simulation with unknown sub-model dynamics. The reasoning for this approach, and its necessity, will be presented. The algorithm will be described in detail, and an application to a simple “toy problem” will be shown. The application of such an algorithm presents a more quantitative way of selecting the time step for co-simulation, and should be understood as a first approach rather than a final solution that is applicable for all cases. The algorithm’s properties will be shown, its limitations pointed out, and possible improvements will be discussed.

Chapter 5 will then close thus text by summing up the presented new approach and its contributions, and by discussing how this work can be continued to provide a more generally applicable algorithm. Possible extensions and applications will be shown.

CHAPTER 2 DYNAMIC MODELS

2.1 Definitions and Basics of Dynamic Models

In order to be able to understand the scope of this work and the method that is proposed in this thesis, it will be necessary to understand the basics of dynamic systems. Of course, within the course of this thesis, only the necessary basics can be explained, but this is deemed necessary to have a complete understanding of the following chapters. Therefore, it will be necessary at this point to deviate from the main text in order to introduce the notion of dynamic systems to the reader. After this deviation, the reader will have the necessary understanding for the development of the following methodology.

In the previous explanations, we have shown that M&S is used in many different applications. For the remainder of this text, it will be assumed that M&S is related to the modeling and simulation of physical dynamic systems. The word "system" describes an entity or whole that is compounded of several sub-parts or members, in the sense of a composition of such sub-parts or members (from the Greek word σύστημ-α, see http://www.perseus.tufts.edu/hopper/text?doc=Perseus%3Atext%3A1999.04.0057%3Aentry%3Dsu%2Fsthma). The system behavior is generally described by a set of rules that govern its behaviors, in particular the spatial trajectories of the system states over time. The states are sets of real numbers which represent points in a state space. The change of states over time is described by the evolution rule. This rule defines the state changes as a function of time. It is deterministic, which means that from a current given state, one and only one particular future state can follow in a given future time. No stochastics (random elements) are involved. The parts of the system generally process inputs and produce outputs, interact with each other, and together form the structure of the whole overall

system. The inputs and outputs are the state points, and can be internal (between the parts) or external (outside the system). Dynamic systems describe systems whose state behavior over time is described using a fixed rule. This implies the time dependency of such systems (hence, dynamic). In fact, this time dependency is of crucial importance to the simulation of dynamic models using digital computers. The fixed rule is commonly an ordinary differential equation (ODE), in the form:

$$\dot{x} = f(x,u,t) \tag{1}$$

This equation describes how the state changes over the next step. The change of the states is the left side of the equation, and it is a function of the current state, potential user or other external inputs, and the simulation time. All variables in the above equation are to be understood as vectors, describing a general state vector setup. This is the most basic form of an ODE. If the changes of states include both time and space, the equation becomes a partial differential equation (PDE).

When trying to determine the state vector development from the fixed rule (the ODE), it is required that the ODE is solved in order to get an equation that is suitable for execution within a digital computer. Oftentimes though, the ODEs are either unknown, or can not be solved in a closed form solution. If the latter is the case, the solution to the equation must be approximated in order to obtain the solution path(s) of the state vector(s). Many algorithms exist to do this within a digital computer, and a particular approach is described further below.

It can be seen that the time of the desired next step is a major contributor to the solution development. More precisely, the time step that is taken from the current to the next point of the solution path is a critical component for the accuracy of the solution,

particularly when using approximate solutions to solve the equation in a digital computer. In general, and in the real world, dynamic systems are executed in real time. This is often referred to as continuous time. A definition of continuous time is given to be the fact that, within a finite time span, the state variables of a system can change their values infinitely often (Cellier, 1991). Figure 6 depicts a notional sample trajectory for a single state of a real world time continuous system.

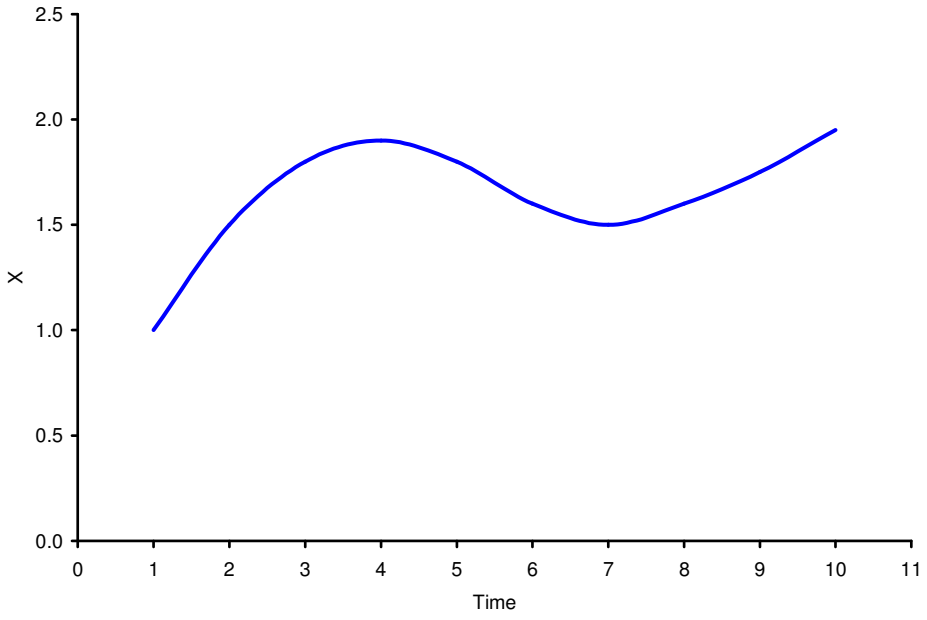

Figure 6. Sample trajectory of a real world continuous time system state

If the underlying exact equation of a dynamic system is unknown, which is the case in most cases, such continuous time systems are represented by sets of differential equations (DEs). Here, two different classes of models can be distinguished. The first, called a lumped parameter model, uses ordinary differential equations (ODEs) to describe the system behavior. The general form of such equations is

$$\dot{x} = f(x,u,t) \tag{2}$$

with x being the state variables, u the inputs into the system, and t the continuous system time. For the special case of linear systems, this equation becomes

$$\dot{x} = Ax + Bu \tag{3}$$

Oftentimes, this is supplemented by the equation for the system output vector y

$$y = Cx + Du \tag{4}$$

Here, A is the state matrix, B is the input matrix, C is the output matrix, and D is the input/output matrix.

The other model to describe continuous time systems is the distributed parameter model, which uses PDEs for system description. An example is the diffusion equation

$$\frac{\partial u}{\partial t} = \sigma \frac{\partial^2 u}{\partial x^2} \tag{5}$$

The other class of models is the discrete time model. For these models, the time is not regarded as continuous, but instead is discretized. Therefore, these models are oftentimes represented through difference equations, rather than differential equations. In the usual case, the discretization steps are equidistant. Figure 7 shows a sample trajectory of a discrete time model output.

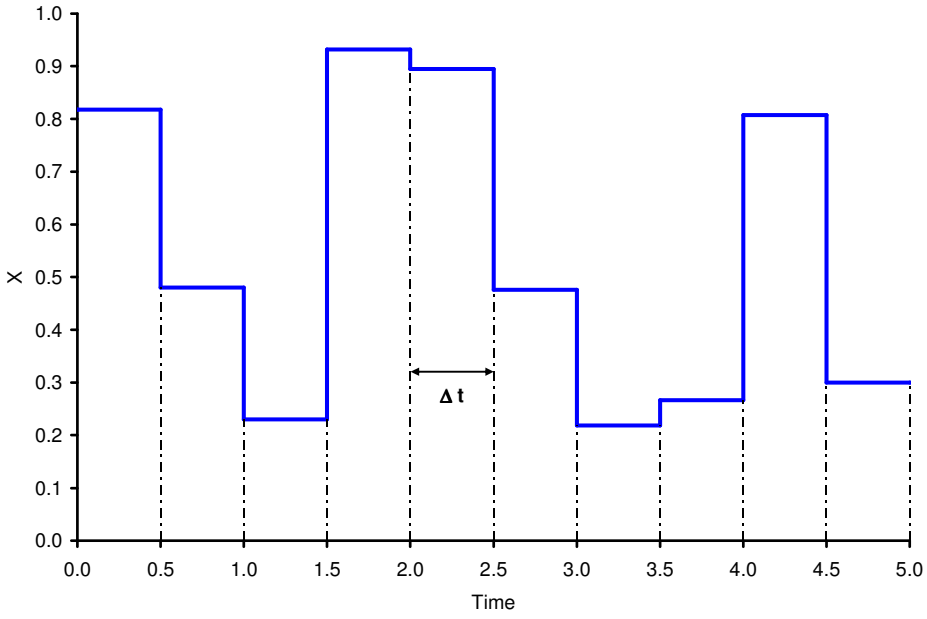

Figure 7. Sample discrete time state trajectory

The difference equations of such models are usually set up as follows:

$$x_{k+1} = f(x_k, u_k, t_k) \tag{6}$$

While most dynamic systems in the real world are in the continuous time realm, and hence would ideally be described with the lumped parameter model for differential equations, such an approach is not feasible when using digital computers to model and simulate such systems. This is due to the fact that the computer has only limited numerical resolution to represent input and output states (and hence does not allow for the infinite resolution of real world states), and due to the property of digital computers that the calculation of the next state will require a certain amount of time. This however, contradicts the observation made earlier, where it was stated that state variables can change their values infinitely often in a finite time. Therefore, in order to be able to simulate continuous time systems on digital computers, it will become necessary (and is also somewhat a "natural" approach) to sub-divide the time axis into discrete, short, and

equidistant (for now) intervals. These intervals are often chosen long enough to allow the computer to compute one set of new state values.

As an alternative to the previously described method, continuous time models can also be "directly" discretized. Using the continuous time state space model from Eq. 1, and apply a discretization step size of Δt (oftentimes also denoted as h), the equation becomes

$$\frac{x_{k+1} - x_k}{\Delta t} \approx f(x_k, u_k, t_k) \tag{7}$$

which can alternatively be written as

$$x_{k+1} \approx x_k + \Delta t * f(x_k, u_k, t_k) \tag{8}$$

This is a discrete time model of the continuous time equation, and would be suitable for use with a digital computer. It has the advantage that it only uses current states to predict the next state output. The time step Δt represents an artificial "clock" in the model that keeps track of the simulation time, and is a metric that can generally be set to an arbitrary value. However, the time step settings will have grave impact on the outcome of a simulation, and discussion on time step settings will follow in subsequent chapters.

The term "time" is easily confused when talking about dynamic continuous time systems. "Time" can refer to the real world time, and would be the choice when describing the facts and relations in real world systems. For example, Eq. 1 would be used world real world time. However, when switching over to the modeling and simulation of such systems, and hence switching to discrete times and time steps due to the application in digital computers, "time" refers to the system time, the time in which the system currently is within the simulation. This is a more abstract definition of time, as it actually does not matter what time the system is in, but rather, what size (order of

magnitude) the time step is towards the next step. This system time must be in line with the system time assumed when the model was created. If the model assumes that one execution step equals one second in real world time, this means that if the simulation assumes one step to be two seconds, there will be a misrepresentation of the results.

To avoid confusion, from here on forward whenever the term “time” is used, it will refer to the system time used in the simulation, unless otherwise noted. Real world time will be referred to with the term “real world time” or equivalents. Where deemed necessary, the terms “real world time” and “system time” are explicitly spelled out.

2.2 A Short Discourse into Discrete Event Simulation and QSS

A more recent development is the QSS (quantized state systems) approach, which deviates from the classical time-discretization approach, towards a discretization of events or event states. Time is kept continuous in this case, and strictly speaking, the event outcomes are also continuous (in that they can have any value unless restricted by the simulation). The input trajectories are piecewise constant functions. The state variables, which are piecewise linear functions, are converted into piecewise constant functions, called state trajectories. See Figure 8 for a sample graph of a discrete event trajectory.

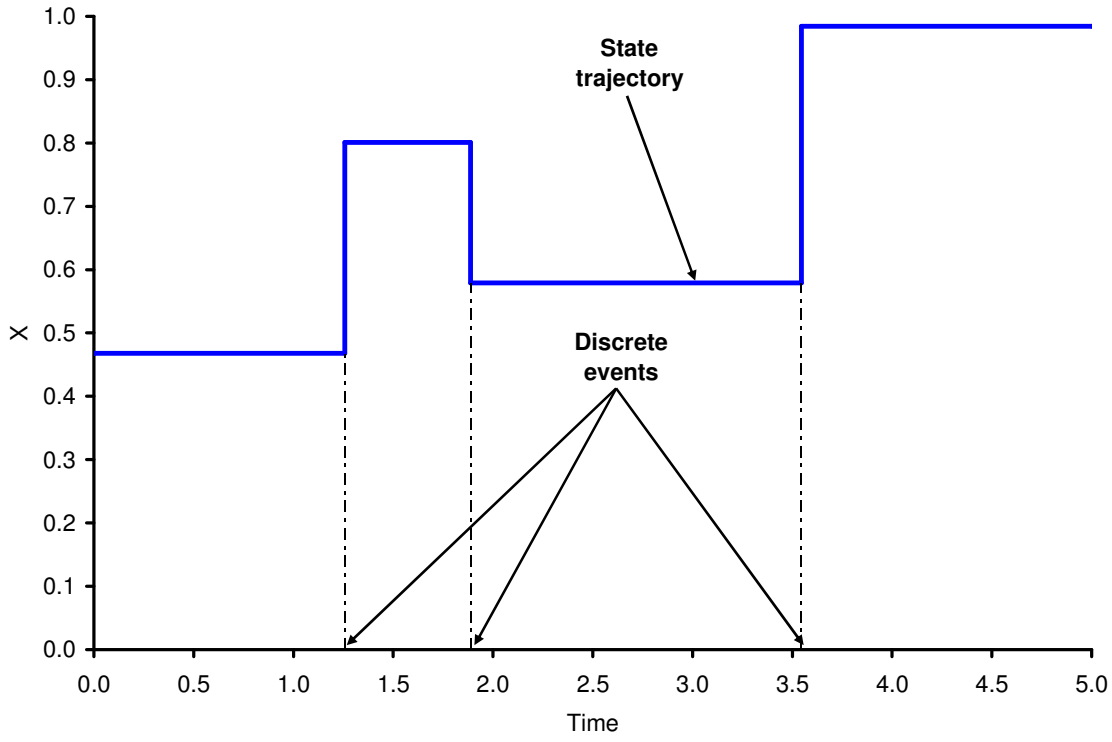

Figure 8. Sample discrete event trajectory

For this, a quantization function with hysteresis is employed. The hysteresis in the quantization function is necessary to prevent the quantized variables of the QSS from performing an infinite number of transitions either at the same time (which does not imply that it is simultaneous), or within a finite time interval greater than zero. The first case commonly occurs in systems with internal transition function cycles, when all cycle states have a time advance function equal to zero. The second case is limited to systems with an infinite state set. If such a system starts from an initial state, and the time advance function is summed up over all the possible successive system states, this sum can converge to a finite value, leading the system to make an infinite number of transitions in a finite time interval. If either of these incidences occurred, it would render the model illegitimate.

Modeling of such systems oftentimes involves statistical and stochastic methods to represent real world behavior. Such methods are for example Markov chains,

Stochastic Petri Nets, Monte Carlo simulations, queuing theory, birth-death processes, etc. Simulation software must be able to handle such methods. Well-known simulation software products are for example Rockwell Arena, XJ Technologies AnyLogic, and various Petri Net programs (see Volovoi (2004) and Volovoi (2006) for references to newer approaches with Stochastic Petri Nets).

One could make the point that strictly speaking, time discrete simulation of continuous time systems is actually discrete event simulation. After all, at a certain event, namely the next time step, the system states change, and those changes in states are continuous. However, the difference is that in discrete event simulation, events are given to the system from external sources. An event might be for example a stock-out (in a logistics simulation) or a simple controls input. The events are independent of the simulation. In continuous time simulation however, the discrete time steps are the events, and are set by the simulation instead of being triggered due to some change within the system; thus they are not independent.

The QSS method has certain advantages over discrete time simulation. The number of calculations can be reduced for a given desired accuracy. It is easier to implement distributed computing, the method is claimed to be able to simulate discrete time models. QSS can be represented and simulated exactly by discrete event simulation. However, due to the inherent properties of this kind of simulation, continuous time systems can only be approximated. The error can be made arbitrarily small by reducing the quantization to zero, provided that certain conditions are met. No error information is obtained during the transients of the continuous system, though. QSS is a relatively new approach to continuous model simulation, and promises a good trade-off between computational expense and accuracy. Its current main applications are in the modeling and simulation of situations that can be more easily and accurately modeled with discrete time events, for example problems in logistics, warehousing, inventory management, etc.

For more information on QSS and discrete event simulation, see e.g. Zeigler and Lee (1998), Zeigler et al. (2000), and Kofman and Junco (2001). However, to the author's knowledge QSS has not been used to co-simulate continuous dynamic sub-systems. This would require in-depth investigations regarding the necessary conditions and inherent limitations of such an approach. Therefore, we focus our efforts on discrete time simulation in this paper. Cellier (1991) provides basic and advanced knowledge on how to model discrete-time continuous dynamic systems.

2.3 Discrete Time Simulation of Continuous Time Dynamic Systems

2.3.1 Basic principles

As argued earlier, the use of digital computers for the simulation of differential equation based time continuous dynamic systems requires a stepwise mode of execution, in which the execution time is split into small steps, the simulation time steps. At the beginning of each step, the system (or rather, its model) is in a particular state, and at a current rate of change for this state (more precisely, for all the state variables under investigation). From this to the following time instance, certain changes in the states will occur, which depend on the current system state, on the model's behavior over time, and on external inputs into the model (such as user inputs). In most instances, the exact path of the state variable is unknown, and hence must be approximated with then information given at the current state. Figure 9 depicts the notional continuous time state path from Figure 6 with discrete time steps.

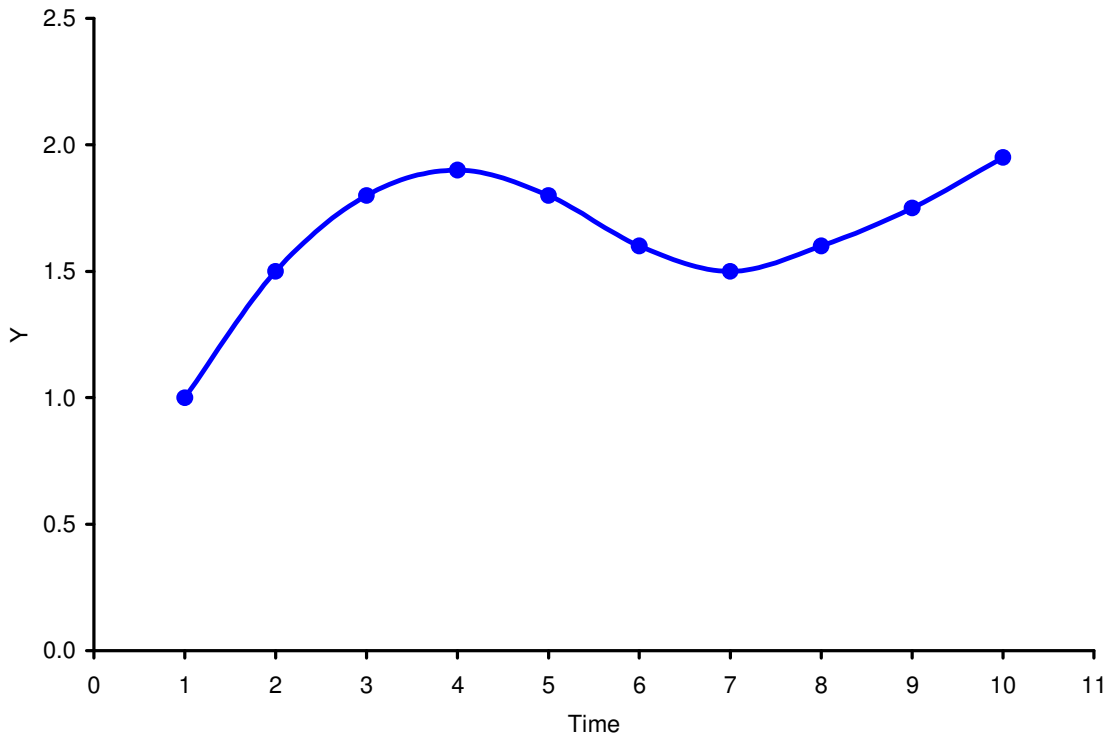

Figure 9. Discretized continuous time state trajectory

A very simple model would be if the output states were defined as a function of the input states in a table that maps every output to an input. This however, assumes discrete states and is therefore not suitable for dynamic systems.

Differential equation models do not specify the next state directly, but rather use a derivative function which specifies the rate of change of the state variables at the current time step. Thus, at a given time, system state and input, all the information available to the system simulator are the current states and the rate of change of those states. Hence, this is the only information we can use to compute the next step(s) of our simulation, to evaluate the new states of the system at the next step. The system equation can therefore not be solve directly, but must be solved numerically, at each discrete step, using numerical integration techniques. This will result in an approximation of the solution. Figure 10 shows the initial state curve, and a nominal approximation solution as it could look like when achieved through numerical approximations.

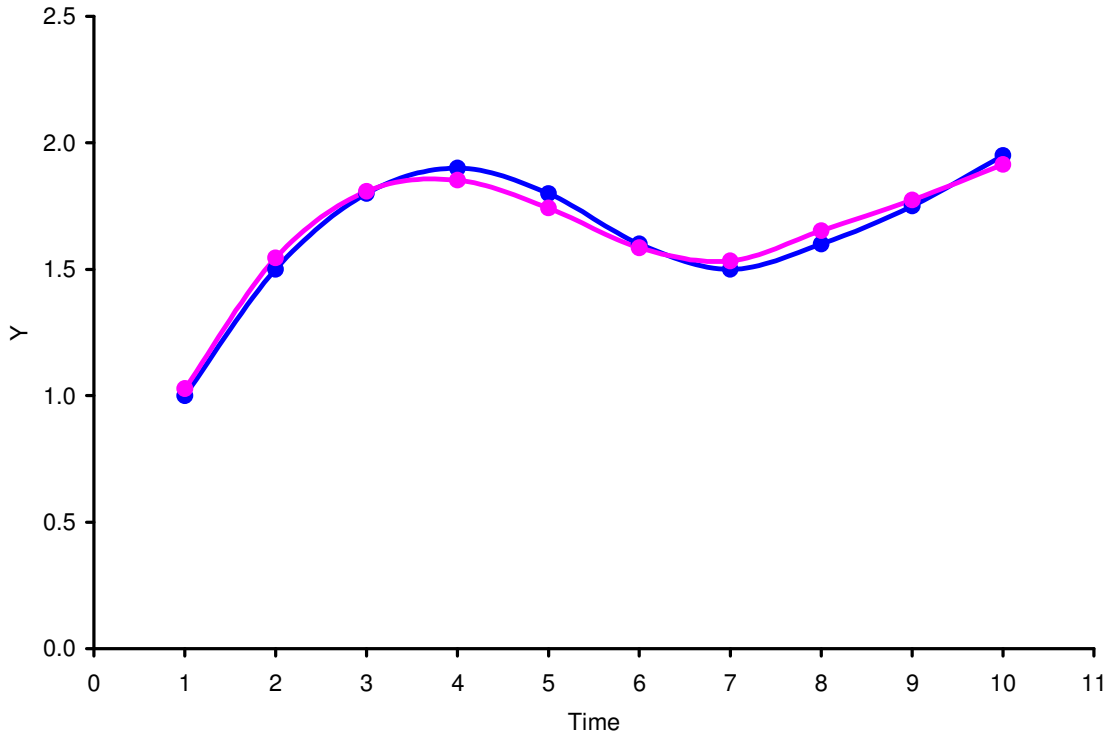

Figure 10. Discretized state trajectory and approximation curve

Most dynamic systems consist of one or more integrators. Simply speaking, this comes due to the form of Eq. 2. The left side is a state derivative. In order to evaluate what this state will be in the next time step, it needs to be integrated numerically, to convert the state derivative in an actual state value. A simple example would be a moving object, such as a planetary lander module. Such a system could be represented by the following sample set of equations (Cellier 1991):

$$\begin{aligned} T &= f(t) \\ \dot{h} &= v \\ \dot{v} &= a \\ a &= (\frac{1}{m}) * (T - m * g) \\ \dot{m} &= -FE * |T| \\ g &= \frac{GC}{(h+r)^2} \end{aligned} \tag{9}$$

This sample set of equations describes the vertical motion of a lunar lander module. Clearly, the three state variables are altitude h, vertical speed v, and mass m. Since there are three state variables, the system is of third order. State derivatives are assumed to be with respect to the independent variable, which for most simulations is (simulation) time. This model also contains several auxiliary variables, which are the vertical acceleration component a, gravity force g, and thrust T, which is an input that varies over time (such as a user control input). The three constants are planet radius r, fuel efficiency FE, and gravitational constant GC. Of course, in order to get a useful, non-arbitrary simulation output, the variables must be initialized before a simulation run.

Looking at the model we can see that some variables (v and a) are used before they have been defined. Normal programming languages and algorithms would create an error here. This however, is explained through the fact that solvers for this kind of equation set treat the equations in parallel. Furthermore, when running such a simulation, initial values for the respective variables are given, which make the equations easily solvable. It is clear to see that the choice of the initial values will determine to course of the simulation run.

2.3.2 Numerical Integration

For later reference in the co-simulation section, the numerical integration of monolithic systems is described first. Here, principles and methods are shown and derived that will later help to understand the issues and solutions for the co-simulation environment and integration problem. In order to get a good understanding of these principles, numerical integration methods are presented here in some detail. These basic integration methods are discussed in some length here, because they build the basis for the algorithms described further below in this text, which will be developed for co-simulation applications. It has been mentioned earlier that a co-simulation model does not

behave like a differential equation and that therefore a strict direct application of DE solving algorithms is not a feasible approach. Nevertheless, since the underlying simulation model will still be dynamic (time dependent), and the sub-models themselves are ODE or PDE models, it can be assumed that the co-simulation model will behave similar to a dynamic model that is described by ODEs or PDEs. While the actual equations are unknown, the underlying approaches towards numerical treatment of the co-simulation state variables follow the same principles as the approaches taken for the development of established ODE/PDE soling algorithms. It is therefore assumed justified to use such established, exact algorithms for the derivation of applicable algorithms for co-simulation.

To solve ODEs numerically, the state derivatives will need to be integrated numerically in order to get the actual values for the states, which are subsequently used to calculate the remaining values. This integration happens at each of the discrete time steps. However, since the time steps are finite in size, the numerical integration must be preformed over that time step size. In order to achieve this, several methods have been developed, some of which are presented in the following.

2.3.2.1 Euler's method

Eq. 8 shows a simple step wise integration algorithm, where the next step depends only on current information about the system states and the current slope. A slight modification of Eq. 8 is as follows:

$$x(t + \Delta t) \approx x(t) + \Delta t * \dot{x}(t) \tag{10}$$

Since all parameters on the right side are known, it is now possible to compute the value of the function at the next time step. This simple approach is commonly referred to as the Euler integration method.

It is however, easy to see that this rule is not free of error. Consider the following figure, which is a zoomed section of Figure 9, see Figure 11.

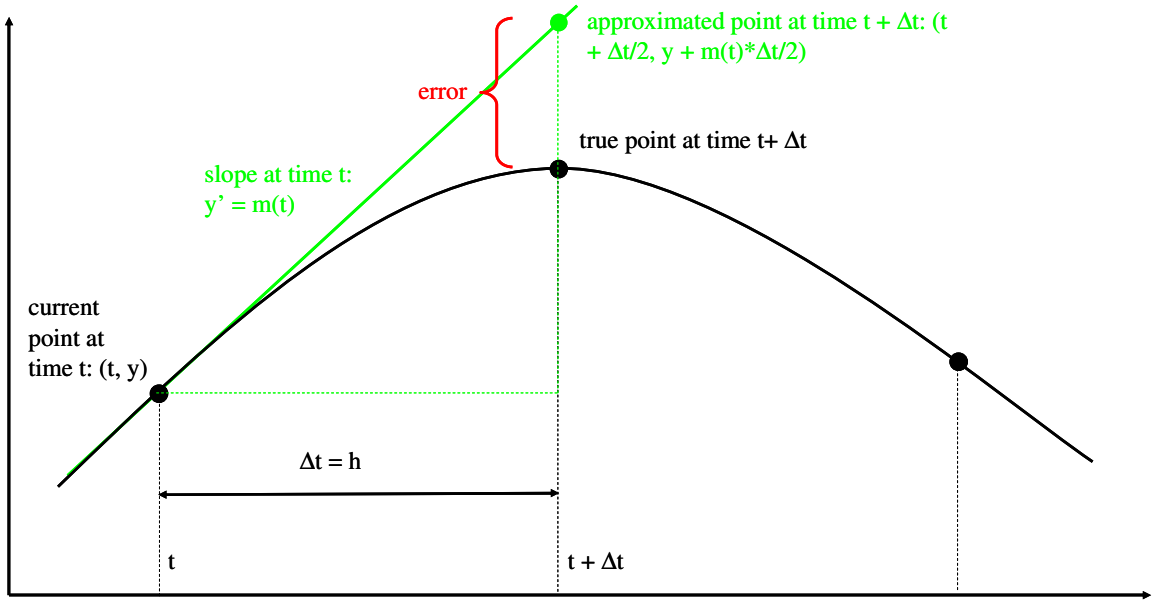

Figure 11. Numerical integration approximation error

We start off at the original point x(t), y(t), which is known (the initial condition). Also known at this point is the slope of the curve, $s(t) = \dot{y}$, and the step size Δt = h. The slope is given by the definition of a differential equation. The time step is set to an arbitrary value, for now. To calculate the next step, we first calculate the new x by adding the step size to x(t), hence we get

$$x_a(t + \Delta t) = x(t) + h \tag{11}$$

Using the slope and the current point, we can also get an approximation for y(t + Δt), which is calculated using

$$y_a(t + \Delta t) = y(t) + h * s(t) = y(t) + h * \dot{y} \tag{12}$$

However, we can see that the new point is not exactly on the original curve, but rather it is off by the difference between the approximate y at the new point, and the

actual result. This error will be positive if the curve is concave and negative if the curve is convex (a curve is convex if its second derivative is positive and concave if its second derivative is negative). This however, is merely a question of definition. The main point is that this kind of integration will introduce a systematic error at the next time step if the slope is not constant. If the slope remains constant, the error will remain constant as well. More generally, it can be stated that as long as the curve is either convex or concave, Euler's method will always produce an error. Also, if the curve does not change its direction (i.e., the sign of the second derivative remains unchanged), the error will cumulate with proceeding system time, unless the second derivative is zero, in which case the error will remain constant. This also means that if the curvature changes its sign, the error will reduce, as it will approach the original curve and eventually cross it. (Note: From here on, the terms "second derivative" and "curvature" are used interchangeably) These crossing points cannot be determined. The error can only give the information as to whether the outcome is too high or too low, compared to what it should be. The total accumulated error itself can be expressed as a linear function proportional to the step size h,

$$\varepsilon \propto c_E * h \tag{13}$$

and a proportionality constant c_E (index E for Euler). Since the step size occurs only as a linear factor, this error is said to be of first order (this order is not related to the order of the underlying differential equation).

An extensive discussion of the error of Euler's method is presented in Gear (1971) and Burden and Faires (2001). General discussion of error propagation in numerical integration is given e.g. by Nakashima (1972). A more specific applied example with specific focus on the round-off error and an actual application solving the heat equation is shown in Lowan (1960). Zadunaisky (1976) describes a method for the estimation of global errors, employing a heuristic condition of validity, to estimate the

propagation of global errors during the numerical solution of ODEs with finite difference formulas. He demonstrates his algorithm with several applications, and claims that the main idea of his method can be extended to other types of applications and to problems solved by Spline functions and PDEs solved by finite differences methods. Choif and Chung (1996) extend the discussion to global and local error estimates and adaptive time stepping for direct time integration in dynamic analysis. They employ a successive quadratic function for the locally exact value of the acceleration variable in their model, and the corresponding parameters for the function are obtained from accelerations at three time stations at every time stage. The local error can then be estimated simply by comparing the solutions obtained by direct time integration method with the locally exact solutions. Based on this local error estimate, they propose an adaptive time stepping technique in a global sense and carry out numerical examples to verify the performance of the procedure. However, their system equations are known, and hence their method is not directly applicable to the co-simulation problem discussed below.

The magnitude of c_E for the Euler method can be evaluated. The following example shows how the error constant can be found for a given underlying equation. Consider the equation

$$\dot{y} = y \tag{14}$$

with the initial value y(0)=1. The solution for Eq. 13 is $y(t) = e^t$, so y(1) is approximately 2.71828. Euler's method gives

$$y_{n+1} = y_n + h * \dot{y}_n = y_n * (1 + h) \tag{15}$$

from which it follows that

$$y_N = (1 + h)^N \tag{16}$$

If $t_N = 1$, $y_N = (1+h)^{1/h}$. the error bound for Euler with $e_0 = 0$ then gives

$$\left|e_N\right| \le h * 2.71828 * (2.71828 - 1) = 4.67077 * h \tag{17}$$

It can easily be seen that especially in the case of high rates of change for the slope (high second derivative), the error can quickly accumulate to significant levels. The question is therefore how the accuracy for this method could be improved. Clearly, one approach could be to use a smaller step size. This discussion will be continued in detail further below, but for now we just say the following: It is possible to reduce the error by reducing the step size. The smaller the step size becomes, the closer the two curves (actual solution, and approximation) will be. There are some implications about reducing the step size, but for now we limit the discussion by pointing out the fact that simply reducing the step size is an inefficient way to increase accuracy, simply because a smaller step size means more steps to be calculated, and hence an increase in computational expense. Therefore, we look into another way to reduce the numerical integration error.

2.3.2.2 Heun's method / Runge-Kutta 2

One method to increase the accuracy is to use a different slope at the current point. The Euler method that was presented here is an explicit method, meaning that it uses only current information of the system to calculate future system states. To solve such methods, the relation

$$Y(t + \Delta t) = F(Y(t)) \tag{18}$$

needs to be solved. Here, Y(t) is the current system state, and Y(t + Δt) is the system state at a later time, after a (small) time step Δt.

In order to be able to improve the slope used at the current position, an implicit method can be employed instead of the explicit method. Such methods find a solution by

solving an equation involving both the current state of the system and a future state that is being estimated by using current state information, namely

$$G(Y(t), Y(t + \Delta t)) = 0 \tag{19}$$

to be able to find Y(t + Δt). Solving this equation requires extra computational efforts, and may be more difficult to implement. These methods are often used to solve equations of stiff systems, which would require impracticably small time steps Δt to keep the error bounded when using explicit methods. Depending on the problem to be solved, implicit method may then be faster and more accurate at larger time steps, even considering the additional computational expense. Hence, the choice of implicit vs. explicit method will depend on the problem under consideration.

How can such an implicit method be employed? Eq. 10 uses the current slope to calculate the next step. From Figure 11 it is clear that a different slope might help to reduce the error at the next step in time. It has been mentioned earlier, that Euler's method will always produce an error if the curve is either convex or concave. If it were possible to find a new slope such that this new slope would bring the next point more closely to the actual curve (ideally: right onto the curve), we could greatly reduce the error of the integration method. To derive such a method, consider Figure 12.

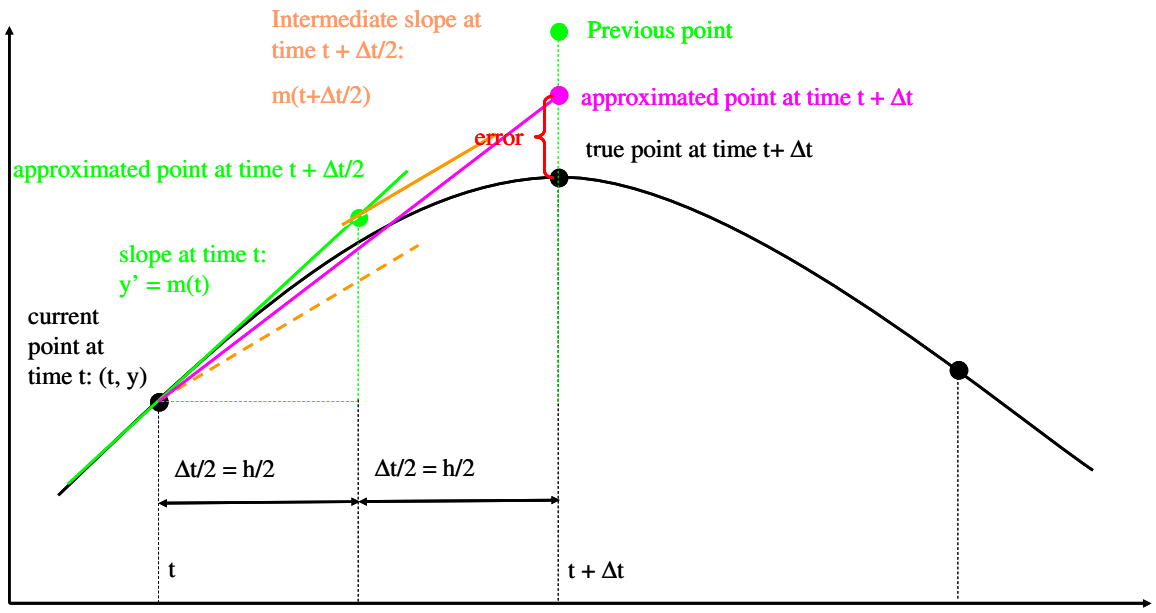

Figure 12. Improved slope derivation

The initial value and function at time t are given as in Figure 11 previously. From this point, the next point is found as described before. However, this new point is only a preliminary reference point, which is located at half the step size of the Euler value, and which will not be used as the actual new point. Instead, for this preliminary point, a new slope is constructed by executing another application of Euler's method with the preliminary slope. The actual slope used to calculate the next step from x(t), y(t) is then the average of the two slopes s(t) and s(t + Δt). In other words, after finding a new slope for time t, the Euler method is then executed with this new slope, for the initial time step Δt. The algorithm thus can be described with the following set of equations:

$$\begin{aligned} x(t+\Delta t) &= x(t) + h \\ y(t+\Delta t) &= y(t) + h * \frac{s(t) + s_{prel}(t+\Delta t)}{2} \\ y_{prel.}(t+\Delta t) &= y(t) + h * s(t) \\ s_{prel}(t+\Delta t) &= f(x(t+\Delta t), y_{prel}(t+\Delta t)) \end{aligned} \tag{20}$$

with y_{prel} being the preliminary reference point at $(t + \Delta t)$, and $s_{prel}(t + \Delta t)$ the preliminary slope of the direction field at $(t + \Delta t)$.

This method is commonly referred to as Heun's method, improved or modified Euler method, or Runge-Kutta 2 (RK2) method. Its total accumulated error is proportional to the step size squared,

$$\varepsilon \propto c_{RK2} * h^2 \tag{21}$$

which is why this method is of second order. c_{RK2} is some constant for the RK2 integration algorithm. This shows that a decrease in step size by half would decrease the error by a factor of four, whereas the Euler method would only decrease the error by a factor of two. This increase in accuracy comes at an additional computational expense, namely the evaluation of the preliminary point. There are even more accurate methods, many under the topic term of Runge-Kutta methods. Most such methods follow a similar concept to the one described previously. However, as has been mentioned before, these methods all rely on calculating various steps ahead of the current time step. Each such step requires the evaluation of the underlying function at the respective time step. This function evaluation is the step that requires the most computational expense. Therefore, the tradeoff here is between simulation execution time and computational accuracy. One of the standard integration methods, the Runge-Kutta method of fourth order (RK4), has an error that is proportional to h^4, but requires four additional function evaluations, which makes it quite inefficient from a computational expense point of view.

It is important to note here that the fixed step Runge-Kutta methods are presented here only because they will lead to methods that introduce adaptive time stepping for numerical integration. They do now however, have a direct implication for the implementation in a co-simulation with adaptive time steps. This is due to the fact that these methods were created to determine the new system state for a given fixed time step,

in order for them to be able to approximate the unknown underlying system equation and the state variable paths. However, this is not needed for the simulation of a dynamic system model in a co-simulation aspect. For such a setup, it is assumed that the simulation will give the right solution, and an approximation of the solution is not needed. Therefore, in the application of an adaptive time stepping algorithm for numerical integration to a co-simulation, the only part of the algorithm that will be applied is the part for determining the time step, not the part for determining the new system state variable values. The rationale for this is the problem to be solved is similar: In both cases, an unknown function for (a) system state(s) must be approximated using the available information about the current, and possibly previous, states and the system behavior. This must be done within certain error bounds that can be attempted to be kept by investigating the system's behaviors and using the time step to adjust the errors. Such algorithms have been developed, and will be introduced below. Their underlying approach will be used to develop an algorithm for the time stepping of co-simulation.

Ideally, a difference approach to the solution of a differential equation could be represented by its actual value at each mesh point, and high accuracy interpolations between those mesh points. While this would be the ideal, there are two sources of error when performing such and approach. The first one is based on the fact that the exact solution of a differential equation is not know in general, and also can not be calculated (it is assumed that the underlying equation has no closed-form solution, otherwise the numerical integration would not be necessary). Therefore, the problem is restated to a form whose solution can actually be found. Since these two solutions are generally not the same, there will be a difference which is commonly referred to as the truncation error. The second source of error is that a real number has an infinite resolution, but can only be represented with finite resolution in a digital computer. This error is commonly referred to as round-off error. When executing the difference method to solve a differential equation, the solution is represented by a finite amount of numbers with finite precision,

which contain both round-off and truncation errors. Therefore, any solution based on a difference method will be inaccurate. However, the inaccuracies may be limited by applying certain algorithms and methods, for example a well-chosen time step Δt, the number of terms in a series expansion, or an improved method for the slope to be used at the current point of evaluation. One goal would be to find out how to best pick those parameters to achieve a desired accuracy while still keeping the algorithm execution time within reasonable boundaries. It may turn out that there is a magnitude of error below which it will be impossible to go. This would then be the maximum possible convergence to the actual problem. For example, if the time step were to be reduced, it follows that the amount of calculations increases. Therefore, the round-off error discussed above would increase, since there are more calculations made. So while accuracy is gained by the time step reduction, it is also traded off by the increased round-off error. This means that there will likely be a point where this trade-off is at a minimum. Generally, an important part of the analysis of any numerical integration method is to study the behavior of the approximation error as a function of the number of evaluations of the underlying function. A method which yields a small error for a small number of evaluations is usually considered superior. Reducing the number of evaluations of the integrand reduces the number of arithmetic operations involved, and therefore reduces the total round-off error. Also, each evaluation takes time, and the underlying function may be arbitrarily complicated.

2.4 Adaptive time step

From the previous discussions on the error of the presented methods, it is also clear that the improved slope is just one way to increase the accuracy of the method. Another other improvement can be made if the integration time step is decreased. As discussed, this will decrease the error of the method, but will also increase the

computational expense, since for every time step there has to be a certain amount of function evaluations. It is therefore desirable to find time steps that represent an "optimum" between accuracy and computational effort. One way of doing this is an adaptive time step. Mihai and Ainsworth (2009) apply an adaptive time step to a monolithic model as a demonstration. The basic idea behind an adaptive time step is that a given tolerance in the deviation of the solution can only be achieved through setting the time step accordingly. An approach that focuses solely on the slope, as described with the fixed step methods above, will give results whose deviation from the real solution will depend on the time step, the general behavior of the curve, the system dynamics, etc., and hence will not be constant or even bounded. Binding the error (deviation) to within a certain error band is only possible through adaptive time step setting.

It can be argued that, if the dynamics of the system are known beforehand, it is not necessary to determine a time step "on the fly" during the simulation execution, but rather determine a fixed time step that reduces the error sufficiently at the beginning of the simulation and use this time step for the whole duration of the simulation run. Such dynamics can be found or at least estimated e.g. by using system identification techniques. However, the problem becomes apparent when changing system dynamics must be expected, which is the case in almost any dynamic system simulation. If such changing dynamics must be expected, then a fixed time step leaves only two options. Either the time step is set sufficiently small to cover even the smallest expected changes in states. Then the expected error is low, but the amount of function calls will not change even if the simulation is in a state of slow dynamics, resulting in inefficient simulation execution. Or the time step is set higher to accommodate the states of slow dynamics. Then the error will increase with faster dynamics, which in extreme cases may lead to inaccurate simulation results or even simulation instabilities. It is in such cases, that an adaptive time step will play out its advantage: Small time steps when the underlying dynamics are fast, thus reducing the error; and larger time steps when the underlying

dynamics are slow, reducing the computational expense at no cost to simulation accuracy. Figure 13, Figure 14, and Figure 15 depict the situation for the three cases discussed above. To realize such a behavior of the system simulation, an adaptive time step is without option.

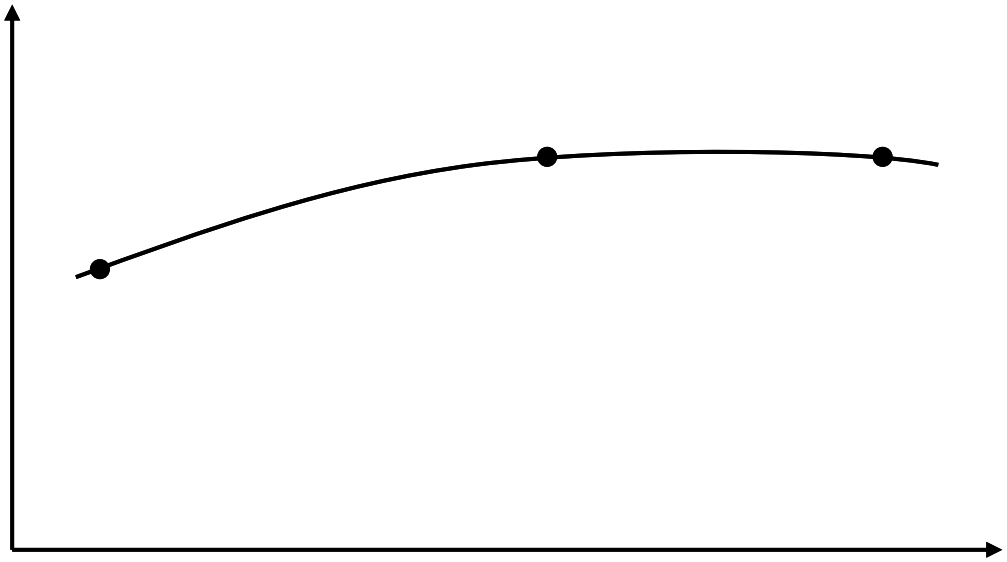

Figure 13. "Slow" dynamics, large fixed time steps

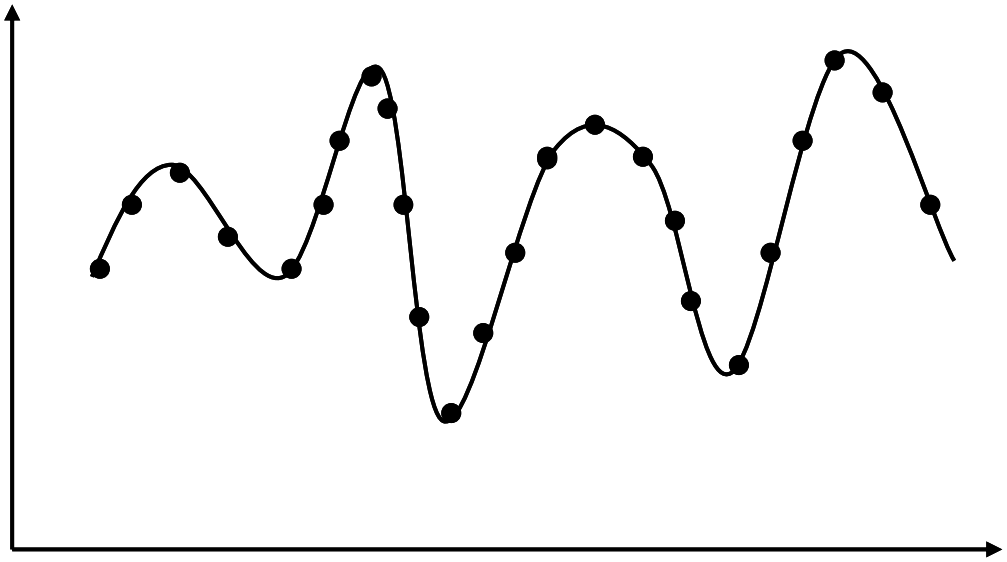

Figure 14. "Fast" dynamics, small fixed time step

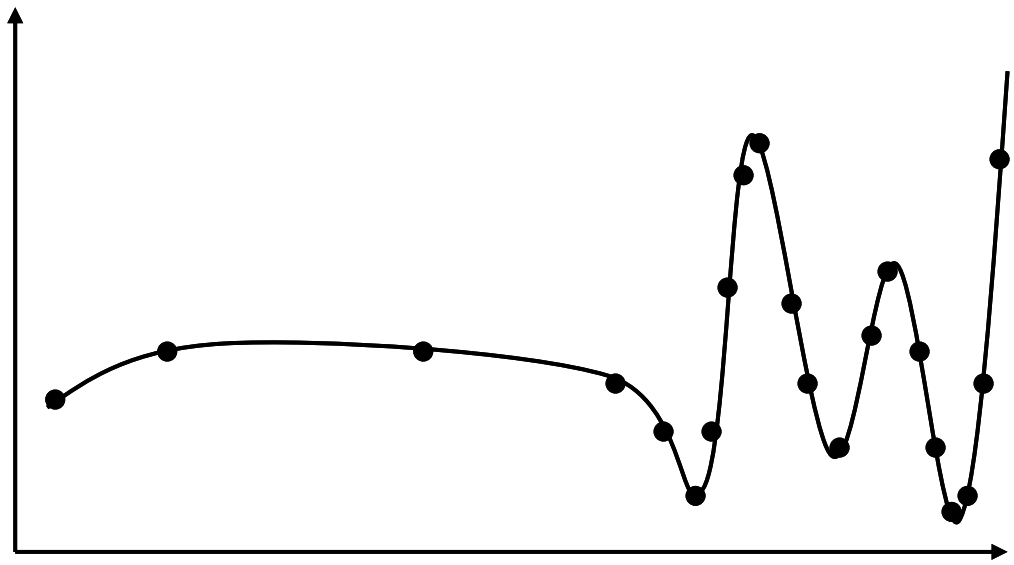

Figure 15. Varying dynamics: Large time steps in regions with "slow" dynamics, small time steps in regions with "fast" dynamics; an adaptive time step is advantageous in this situation, and required if error bounding and limitation is necessary

2.4.1 Historical Development Of Time Stepping Methods

The idea of using adaptive time steps for numerically solving differential equations in system dynamics goes back several decades. The most famous methods for numerically solving ordinary differential equations (ODEs) with constant time steps were formulated in the early 1900s by German mathematicians C. Runge and M.W. Kutta, and are generally referred to as Runge-Kutta methods. Those methods, no matter their order, are fixed time step methods, and improve the solution approach only through derivative (slope) approximation improvements. Fehlberg (1969) extended these methods by adding an adaptive time step algorithm. A first application was for heat transfer problems (Fehlberg, 1970). The idea behind this approach was to be able to control for the integration error by adapting the time step such that the error remains within certain prescribed error bounds. The general approach is a predictor-corrector approach. A first

predictor solution is calculated using N future points from the current location. Then using one additional point, a corrector is calculated. Based on a prescribed error, the time step is then set such that the error will remain below the prescribed error. It is an iterative approach, which means that the algorithm is repeated until the error bound is not violated any more. Such and algorithm is usually denoted by its two orders, namely the number of predictor and corrector points. For example, an RKF45 indicates a Runge-Kutta-Fehlberg algorithm with 4 predictor and a 5th corrector point. This was a first, and by today's means almost primitive approach towards the problem of finding adaptive time steps, but it nevertheless served the purpose well. These algorithms are still widely used today, even in standard mathematical software packages. They have however, also been the basis for numerous extensions and improvements. Krogh (1973) already lists approximately 10 different adaptive step size algorithms and their advantages and disadvantages. A specific emphasis is made towards time step halving and doubling in order to find a "best" time step for the method. The inherent instabilities created by the halving process are discussed, and a method for curing them is proposed. Gupta and Wallace (Gupta and Wallace, 1979) extend the methods of adaptive time steps to higher order methods up to order 14, and compare their algorithm to previous developments and fixed step algorithms. They find that adaptive time steps may not always be superior to fixed steps, and that their current algorithm is not inherently stable at all times.

An algorithm for adaptive time stepping with particular focus on dynamic system problems is introduced by Bergan and Mollestad (Bergan and Mollestad, 1985). The authors describe an approach for using objective criteria for performance and guidelines for developing practically applicable algorithms. They also introduce a set of parameters

for the characterization of the dynamic response of a system, including parameters for current frequency, current period, and dynamic stiffness. Based on the current period, an algorithm for automatic computation of the time step is outlined, and aspects of adapting this algorithm to special types of applications are discussed, including the use of a 'tuning function'. Ideal requirements for time-stepping algorithms are discussed with respect to general criteria and practical aspects. A special application of a variable time stepping algorithm is presented in Nakano et al. (1993). It deals with molecular simulation on parallel computers. Molecular simulation is one of the fields where numerical simulation is used heavily. Simulating molecules is very similar to the basic N-body problem in mechanics, and a standard case for numerical integration simulation. Due to the high number of molecules (bodies), parallel computation is the method of choice for simulating and solving such systems. The authors use multiple time steps for the distribution onto several computer nodes, and calculate the multi-body interactions using an adaptive time stepping (MTS) algorithm in order to be able to increase the algorithm efficiency and reduce computational expenses.

A similar problem is discussed in Saha and Tremaine (1994), only in other orders of magnitude: the authors use individual time steps for different planets to investigate planetary movements on a long term time scale. An application for adaptive time stepping in dynamic robot systems is presented in Joukhadar and Laugier (1996). Their work describes an adaptive time step approach to dynamical simulation based on monitoring the conservation of energy. Their approach ensures numerical stability and computational efficiency at a very low computational overhead, and provides estimates of the probable range of numerical errors in position and velocity of a robot arm. The

approach is applicable to any physics based dynamic simulation system and can be integrated with optimization techniques. Cardenal et al. (1999) propose a time stepping algorithm that uses the index of the augmented Lagrangian formulation of the equations as its time step changing parameter. The example is limited to the integration of the equations of motion of constrained multi-body systems in descriptor form. The method takes advantage of the better performance of the index-3 formulation for large time steps and of the stability of the index-1 for low time steps, and automatically switches from one method to the other depending on the required accuracy and values of the time step. The authors claim that the use of the proposed technique allows a substantial speedup gain in CPU time for large scale systems, thus helping to achieve real-time behavior, and that the method is general and can also be applied to solve the dynamics of flexible multi-bodies. An error estimate for one step implicit time stepping schemes of a special kind (the Pade type, which is commonly used in structural mechanics) is given in Ruge (1999). The time step here is calculated based on an error estimate based on a given local accuracy, using an a priori error estimation for one step implicit methods which allows finding a nearly optimal step size before solving the transition equation for the actual time step.

Anitescu and Potra (2002) extend the multi-body problem and present a time-stepping method for rigid multi-body dynamics with contact, friction and stiff external forces. The multi-body simulation case is a standard and widely applied basic case for investigations of parallel and co-simulated, multi-time and multi-space models. The author's method is well defined for sufficiently small time steps and is unconditionally consistent for the case where the stiff force originates in springs and dampers attached between two points of the system. Their time stepping approach is especially helpful for

stiff systems, and subsequently considers integration step approaches particularly in stiff limit cases. Lopes and Bermudez (2001) develop a general method for designing and evaluating time stepping algorithms. Their approach is to use a know time stepping behavior to create a learning plane, from which future time step setting can be extracted when similar system state situations occur. This is somewhat similar to training an algorithm for a specific behavior under the condition that similar behaviors have been observed in sample runs before. The learning plane combines the evolutions of both the step size and the mean square error, includes both transient and steady-state behaviors, and can be used to compare performances of different algorithms against an optimum trajectory in the learning plane. In system identification applications, the algorithm can be employed to optimize a time stepping algorithm. An application of Runge-Kutta methods for multi-rate partitioned systems of ODEs is presented in Günther et al. (2001). It specifically deals with the coupling of subsystems in a hierarchical modeling approach. In order to cope with stiff problems, the authors introduce multi-rate schemes based on partitioned Runge-Kutta and multi-rate Rosenbrock-Wanner methods which avoid the coupling between active and latent components based on interpolating and extrapolating state variables. They demonstrate the algorithm by employing an MPRK2(3) (Multirate Partitioned Runge-Kutta of order 2(3)) method on a sample problem. They conclude the effectiveness of MPRK methods, provided that the stiffness of the overall problem is not extremely large, in which case implicit methods should be chosen at the cost of higher computational expenses. De Kok and Wind (2002) present a method for estimating the appropriate time step for a model in an integrated systems network. It pays tribute to the fact that in modern computer simulations, more and more models will need to be

integrated into an overall system and an integrated systems network. For this case, the problem of time step setting becomes less trivial and more mathematically involved. In general, such problems need to pay more attention to spatial and temporal detail. The specific example which is used in the paper is the WadBOS model which has been developed as a case study for a decision support system for the Dutch Wadden Sea. The model describes the interaction between e.g. recreational navigation, cockle fisheries, and military activities with the natural functions that exist in the Wadden Sea. The paper's approach is based on a comparison of the intrinsic model uncertainties resulting from parameter distributions and the input noise from interacting models with the error induced by the numerical procedure used to solve the difference equation. The example discussed pertains to a single model for biomass growth, but the authors claim that it can be generalized to integrated model networks. The authors also realize the limitation of the method, which is that it requires calculation of the functional derivatives for each system equation, which results in the required computations to become more intensive for chains of interacting models. Nevertheless, this is a first application of a time stepping method for integrated systems.

Jansson and Logg (2004) introduce an algorithm for multi-adaptive Galerkin methods in the context of dynamic systems. An extended mass spring system is used as the underlying model, and a multi-adaptive method is used to integrate the mechanical system using individual time steps for the different components of the system while adapting the time steps to the different time scales of the system. A "toy" problem of two masses, connected through a spring, is used to demonstrate the effectiveness of using individual time steps for each component. The method is then extended to a multi-body

problem with multiple masses connected through multiple springs in a three-dimensional space. The authors show that multi-adaptive methods outperform mono adaptive methods for systems containing different time scales if there is a significant separation of the time scales and if the fast time scales are localized to a relatively small part of the system, and that multi-adaptive time stepping, and in particular the presented implementation, works in practice for large and realistic problems. This paper draws on findings presented in Logg (2004). The algorithm is further refined and applied to practical examples in a follow up paper by the same authors (Jansson and Logg, 2008). Time integration of ODEs with required resolution of the fastest time scales of the system can be very costly if the system exhibits multiple time scales of different magnitudes. If the different time scales are localized to different components, corresponding to localization in space for a PDE, efficient time integration requires the use of different time steps for different components. Multi-adaptive Galerkin methods select the time step sequence individually and adaptively for each component, based on an a posteriori error estimate of the global error. The multi-adaptive methods require the solution of large systems of nonlinear algebraic equations which are solved using explicit-type iterative solvers (fixed point iteration). If the system is stiff, these iterations may fail to converge, corresponding to the well-known fact that standard explicit methods are inefficient for stiff systems. To resolve this problem, the author presents an adaptive strategy for explicit time integration of stiff ODEs, in which the explicit method is adaptively stabilized by a small number of small time steps.

An adaptive numerical time stepping algorithm that is capable of detecting when the system glides into chaos is presented by Chen et al. (2007). Their scheme has a form

similar to the basic Euler fixed step numerical integration scheme, but the step size is adapted automatically. Their algorithm can forecast the appearance of chaos if the considered dynamical system becomes chaotic. While this is a side issue of the general adaptive time stepping problem, it nevertheless offers some insights into the limitations of time stepping and the necessary attention that must be paid to the underlying system behaviors when determining time steps. Another problem that frequently occurs in the simulation of dynamic systems is that of discrete events, or "switches", such as discontinuities, jumps, discrete variables, slip-stick-changes and similar. Under such conditions, classical time stepping algorithms may work poorly due to the abrupt changes in the system.

Pfau (2007) introduces a modification to the classical algorithms that takes a priori information about the underlying system into account to properly react to such discrete events within the continuous system simulation. The proposed algorithm leads to a decrease in the number of failed integration steps and to more efficiency in the integration algorithms for systems with "switches" and the usage of a priori information for the step size control. Another application of a multi-time scale algorithm is presented in Mahjoubi et al. (2008), with a specific emphasis on structural dynamics. In structural dynamics, commonly the structural domain is discretized in space with finite elements and a single numerical time integration scheme is used with a single time-step for the entire domain. Using a uniform time step for the entire mesh to meet stability and accuracy requirements is computationally very inefficient. Therefore, the structural mesh is subdivided into smaller sub-domains, which allows setting the time step according to the individual requirements of each of the sub-domains. The sub-domains are created

using a dual Schur type decomposition (also called FETI method, for Finite Element, Tearing and Interconnecting). This is a commonly employed method for domain decomposition, which is efficient and practical for the parallel solution of for linear first order transient partial differential equations (PDEs), the type of equations commonly found in systems with both spatial and time domain discretization. The coupling is achieved through Lagrangian multipliers, and adjusted such that velocity constraints at the domain borders are satisfied. The authors offer propose a general formalism for a wide range of time numerical schemes which enables to couple sub-domains with their own time integration scheme with large ratio of time scales. This work is based on a previous publication by some of the authors (Combescure and Gravouil, 2001).

Shin and West (2008) introduce a new class of time integration methods designed for efficient simulation of multi-scale ODEs and PDEs, which they refer to as Multistep Asynchronous Splitting Integrators (MASI). They are extensions of the previously developed methods Asynchronous Splitting Methods (ASM, see Lew et al., 2004) and Asynchronous Variational Integrators (AVI, see Lew et al., 2003). The MASI algorithm determines time steps for the different sub-domains, and applies a specific time stepping scheme to the execution of those sub-domains. Namely, the MASI algorithm will ensure that at any stage the least advanced component of the vector field (the vectors of the states of the underlying equations) is time stepped next, which defines an asynchronous ordering of operations. The authors provide numerical evidence to show that for multi-scale systems with significant scale separation, high order MASI methods are cheaper on a cost versus error basis than either low order asynchronous methods or high order synchronous multistep methods, and that the improvement achievable by MASI methods

increases with increasing scale separation. Hutchinson (2009) uses a time stepping scheme for meteorological investigations in weather forecasting. The time stepping scheme presented is based on the Courant number (a metric used in numerical simulation of fluid flows for the discretization of time dependent partial differential equations. It determines by how many "cells" a metric under investigation can move in one time step), and the algorithm directly determines a time step size depending on that number. Since the courant number is only applicable in certain partial differential equations (PDEs), this solution is only applicable under such conditions, and thus very specific.

The application of a time stepping scheme for non-smooth systems is discussed in Acary (2009). Smoothness of a dynamic system behavior is usually a prerequisite for general numerical integration algorithms to work, particularly when adaptive time stepping is involved. A non-smooth event, as discussed in the paper, is for example a bouncing ball. If this issue is not addressed correctly, the solution at the non-smooth location(s) will become inaccurate. This paper presents and compares two time stepping schemes, Moreau's sweeping process and time stepping scheme, and the Schatzman Paoli time stepping scheme. The paper derives local order estimates for both schemes, and general estimates for the local error. It discusses a continuous Lagrange multiplier with a single impact in the time step, and investigates how the time step can be set in the case of non smooth situations by proposing a way to evaluate the error in such a case. A variable order approach is also proposed. The author is aware that his algorithm behaves well on simple academic examples, and that it has to be improved on more complex nonlinear multi-body systems.

For complex system integration simulation setups, where a variety of collaborative disciplinary subsystems for time-synchronized running are handled dynamically in a distributed environment, Zhang et al. (2010) discuss the handling of local and global time steps and their synchronization within such an integrated environment during run time. They describe how a time stepping algorithm that updated the time step individually for each sub-system will inevitably create the situation where the time steps between the sub-systems is not synchronized because there is no strict rate ratio between the sub-systems. Hence, the algorithm must be a generalized multi-rate algorithm. The necessary data points for the sub-systems to use with interaction between the systems must be acquired through interpolation between the subsequent points in each sub-model, and passed along between the sub-models during simulation execution. However, this method does not propose or use an overall time step for data exchange synchronization between the sub-models.

Commonly, the algorithms for numerical integration of ODEs and PDEs use the first derivative (slope) as a basis for the approximation of the solution and the setting of an appropriate time step. Subsequently, they are first order methods. However, there are more sophisticated methods that use higher order derivatives, as well as extrapolation methods for determining the time step setting and solution to a numerical integration solution. Kulikov and Khrustaleva (2010) present an approach for higher order derivative Runge-Kutta methods with automatic step size and order control. The algorithm is based on an earlier work by the authors (Kulikov and Khrustaleva, 2008), and attains high convergence rates and accuracy were due to an automatic selection of the optimal step size and order at each grid point. The computational methodology presented has a

practical potential provided that the calculation of the derivatives is not too expensive. A simplified Newton iteration proposed in this paper makes it possible to substantially reduce the computation time due to the reduction in the number of arithmetic operations performed at each step of Newton's method while still preserving high convergence rates. An algorithm that allows for increased efficiency of a self-adjusting multi-rate method is introduced in Savcenco and Mattheji (2010). Based on the Rosenbrock methods as their basic numerical integration methods, the multi-rate time stepping strategy is described. The algorithm computes a first, tentative approximation at the new time level for all components. For those components for which the error estimator indicates that the local temporal error is larger than a given tolerance, the computation is redone with smaller steps. The refinement is recursively continued until the error estimator is below the given tolerance for all components. The improvement is done by deviating from the original's recursive halving of time steps towards a direct calculation of a smaller time step for fast components. Similarly, for fast components, the time step is not doubled subsequently as in the original method, but rather adapted in steps larger than a factor of 2. The paper shows that the efficiency of time integration methods can be significantly improved by using large time steps for inactive components, without sacrificing accuracy.

As can be seen by this small excerpt of the literature, the adaptive time stepping technique is widely discussed and applied in various instances. Its application ranges from the solution of simple ODEs and systems of ODEs, to stiff systems, PDEs, and split systems with different time scales in the sub-domains. However, the literature shows a lack of treatment with respect to the issue of adaptive time stepping in co-simulation with unknown mathematical system description ("Black Boxes"). All the above papers

essentially assume the knowledge of the underlying system equations, based on which a mathematical treatment of the time stepping problem is executed. Based on these assumptions and information, some methods that have been mentioned above, e.g. Galerkin methods, Lagrangian multipliers, eigenvalues, etc. are made possible. In particular, the solutions assume the knowledge of the state derivatives, either by knowing the underlying differential equations or by obtaining the derivatives as outputs of the models and sub-models. Since most solvers and subsequent methods are based on the numerical solution of differential equations, these derivatives are essential for the numerical solution of the equations, and usually given. This is valid for single equations as well as more complicated sets of equations and even co-simulated systems. The problem identified is that none of the methods described here are suitable for direct implementation in a co-simulation environment where the underlying mathematical equations, and particularly the state derivatives, are unknown. As will be discussed later, this might be the case even if the underlying sub-models of the co-simulation setup were created by in-house engineers: Not all modeling software will automatically generate the system equations and make them readily available for the integration engineer. Under such circumstances, the traditional algorithms can not be applied, at least not directly, because they all rely on the equation of the systems and sub-systems to be available. Nevertheless, the methods discussed in the literature can be adapted to the case where equations are not available. This thesis is an attempt to adapt such an algorithm to a co-simulation problem where the underlying equations are unknown, and provide a first approach towards solving the problem of requiring the system equations.

2.4.2 Time-Stepping Algorithms And The Runge_Kutta-Fehlberg 2/3 (RKF23) method

Any method that determines a time step for the current system state and the next step to be taken must do so by estimating the behavior of the system during the following time period. This means that all the adaptive time stepping methods calculate future points and from these results derive the system behavior and an accurate time step. The basic integration methods, such as Euler, RK2, RK4, etc. all calculate future time steps as well, but use this information for an improved slope at the current point in time, instead of for the calculation of an adaptive time step. However, over the years these algorithms have been developed further to be able to accommodate an adaptive time step and thus a control of the error as well. This is the reason why this thesis describes the functionality of numerical integration schemes: These algorithms form the basis of the adaptive time stepping algorithms that are used in the co-simulation setup in a following chapter. The reason for this approach is that the adaptive time stepping schemes for numerical integration algorithms are well developed and proven, and have a solid mathematical foundation. It is therefore not a completely new territory that has to be entered and explored when adapting such algorithms to a new problem, namely that of co-simulation time stepping.

The underlying principle of the widely applied Runge-Kutta-Fehlberg 4/5 method (RKF45) and other adaptive time stepping methods is that they take certain points in the future and derive a time step setting for the next step. Before taking this step however, they calculate one additional point in the future, and check their preliminary time step result against this additional information. The first part, calculating several points in the

future and deriving preliminary time step information, is commonly referred to as *prediction*. Calculating an additional point, and adapting the preliminary time step against this additional information, is called *correction*. Hence, such methods are commonly referred to as *predictor-corrector* algorithms. These methods allow for error bounding, and determine "optimal" time step for given error. However, these methods share the same shortcoming that all the basic methods also had: Their calculation of a "best" time step for the current point in time is based on multiple future points, which translates into multiple function evaluations. RKF45 for example, calculates four points for prediction, and a fifth point for correction. Hence, for every time step taken by the simulation, the model must be evaluated at least five times. Since the algorithm is of an iterative nature, the steps might have to be repeated if the current time step calculated turns out to be unfit for the error binding to the given tolerance level. This will increase the computational expense even more. Of course, these calculations involve *all* variables and states of the model. For complicated and complex models, this will very likely result in large real world time requirements for appropriate simulation results. Figure 16 shows a notional graph to depict how the order of magnitude of function calls (model evaluations) varies with given error tolerance.

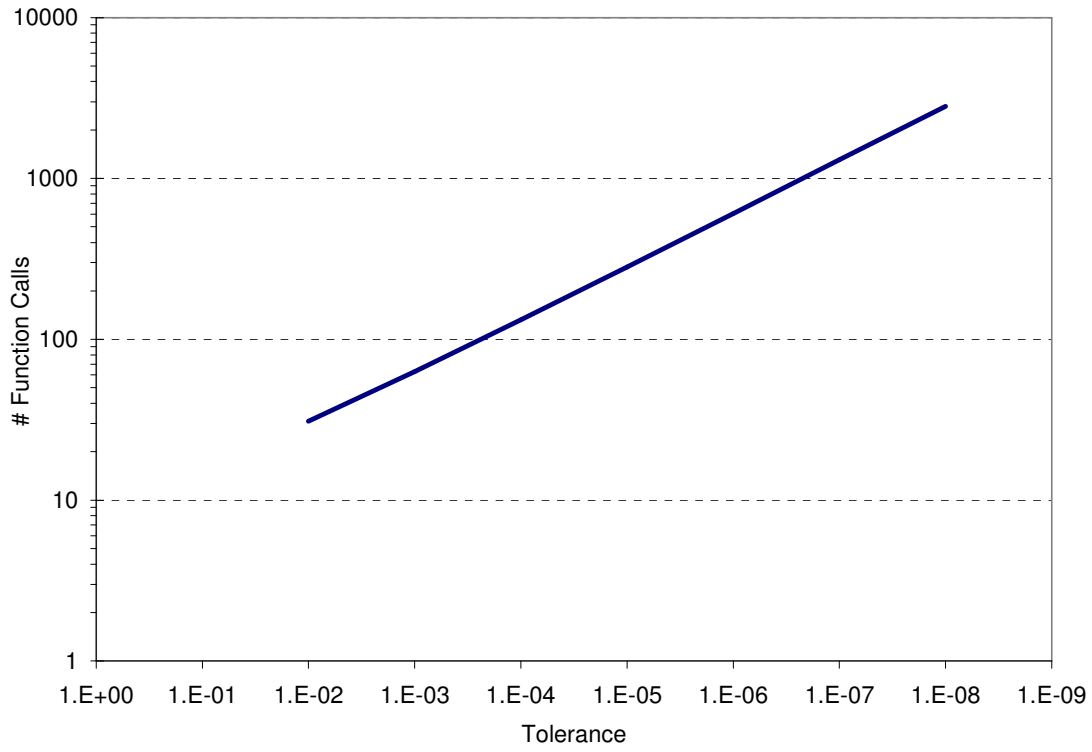

Figure 16. Notional graph of number of function calls vs. desired tolerance, RKF45 algorithm

Therefore, a modified RKF algorithm was investigated, which allows for adaptive time step and error control but which does so with fewer function calls and subsequently less computational expenses. This algorithm, the Runge-Kutta-Fehlberg 2/3 (RKF23) algorithm, has proven to be sufficiently accurate at acceptable computational expense. It is a good compromise solution to the RKF45 algorithm. It will be described in more detail here, and adopted as the time step setting algorithm used in co-simulation in Chapter 4. The reason why such an algorithm is proposed to be used for "Black Box" co-simulation is that the problem that the algorithm has to solve is very similar to that which it has been developed for in the first place: the algorithm needs to be able to accurately follow the path(s) of (a) state variable(s) that is unknown. Just like in the case where the unknown path of an ODE state has to be approximated within a given error bound in the most basic case and through numerical integration algorithms (similar to Figure 6), the

state paths of the co-simulation variables exchanged at every global time step need to be kept within a certain prescribed error bound to the assumed “ideal” path. Thus, the problems to be solved are similar: The unknown state path(s) for (a) system variable(s) must be approximated by using information about the current (and possible previous) states and the behavior of the system state(s). Algorithms to do this exist for numerical integration of differential equations. This justifies the use of these proven algorithms as a first approach for the solution of the time stepping problem.

Similar to the RKF45 algorithm used above, the RKF23 algorithm is an algorithm that uses the slopes of a given ordinary differential equation (ODE) to approximate the path of the underlying variable values that is described by the ODE. It is an algorithm of the predictor-corrector type. This means that it will calculate a future time step as a prediction values, followed by another future time step that is used as a corrector value. Unlike the RKF45 algorithm, RKF23 uses only 1 point for prediction and 1 point for correction, resulting in a total of three function calls required per simulation time step. Based on these two values and their relation, a new time step is established and a new point on the approximated path of the underlying variable values is calculated. This enables the algorithm to incorporate a means of error control.

The actual approach of the RKF23 algorithm is as follows. From the current point, a future point is taken at a distance half of the current time step. For an ODE, this means the algorithm considers the slope at this future point. With this slope, it calculates the future point at a distance equal to the current time step from the current point. This is the predictor point. In a second step, slopes at the current point, the point at half of the current time step (half-step point), and at the full time step (full-step point) are taken, and a new slope is calculated that is used to establish a new point at a distance of one current time step to the current point. The weights for the three slopes used are 1/6, 4/6, and 1/6, respectively. The new point is the corrector point. The error is calculated by taking the difference between the predictor and the corrector point. If this difference is greater than

a predetermined error tolerance, then the algorithm reduces the time step according to a specific formula, and repeats the approach. Otherwise, the new point is being used, and a new time step is determined for the next integration time step. A meta code for the RKF23 algorithm is described in Appendix A. Figure 17 and Figure 18 depict the different stages of the RKF23 algorithm, Figure 19 shows how the new time step is then calculated.

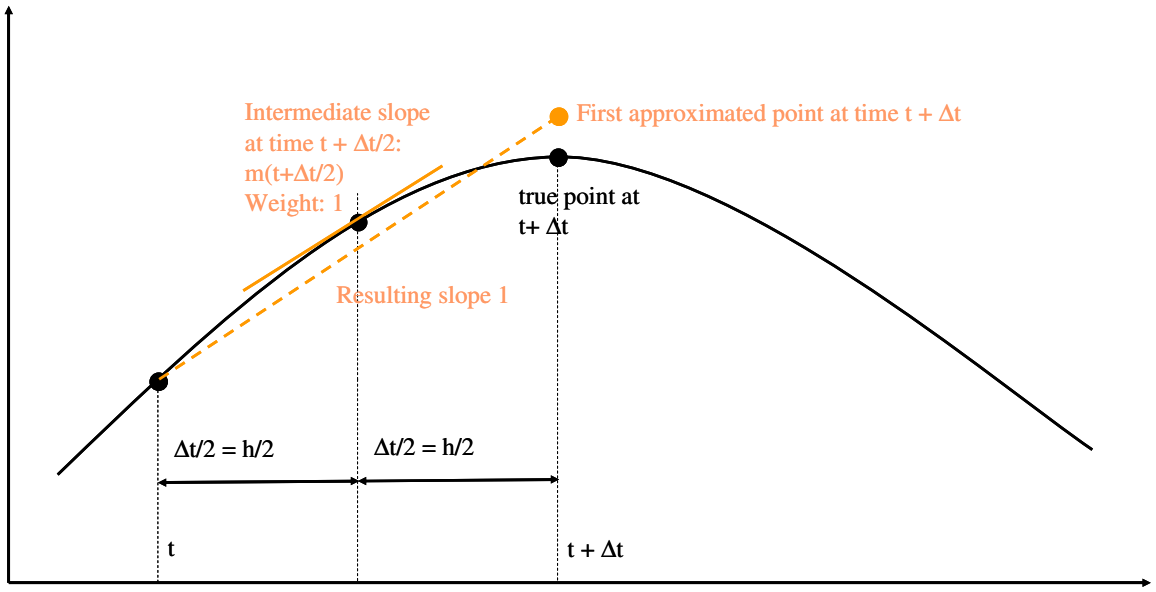

Figure 17. Predictor step: Use Δt/2 slope to determine new point at T + ΔT

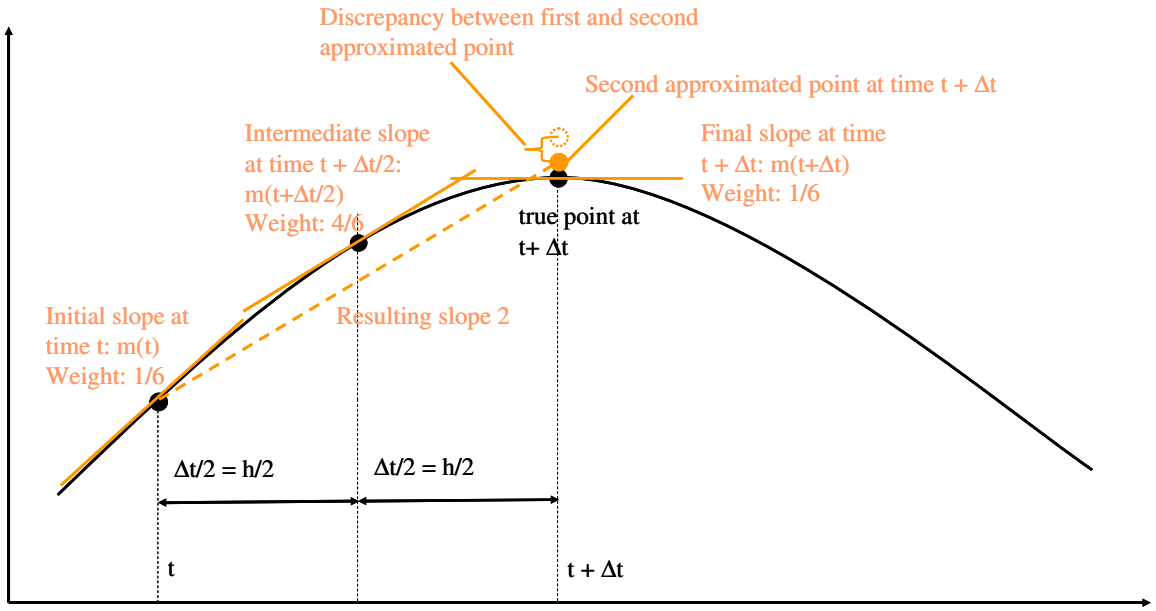

Figure 18. Corrector step: Use current, ΔT/2, and ΔT slopes to determine new point at T + ΔT

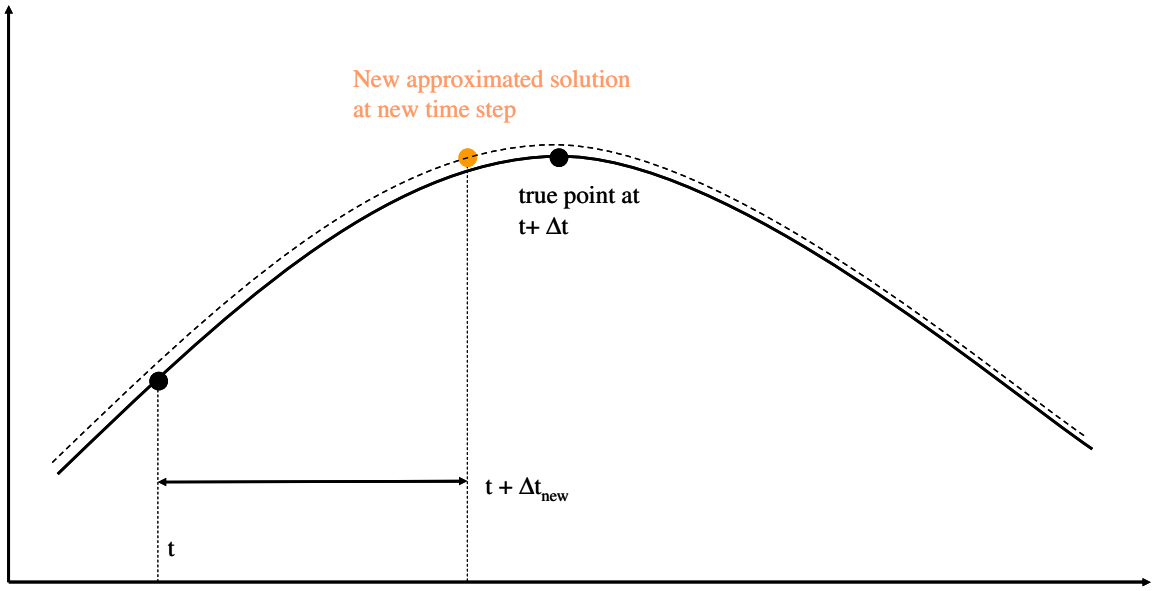

Figure 19. New point and time step determination: Use error / discrepancy between predictor and corrector points to determine new time step; set new point at T + ΔT_{new}

It can be readily observed from the above meta code, that the time step is changed even if the current time step was established to be sufficient to use it for the calculation of

the next point. This behavior makes the RKF23 algorithm iterative. This is the reason why any algorithm that uses current and previous data only will not be able to succeed in correctly determining the time step for the next simulation step.

This algorithm was now implemented in Matlab, and sample curves were used to test the algorithm's applicability and accuracy. The sample curve was chosen to behave similar to the simulation model with which it would be tested later on in the actual co-simulation. The sample curve was also parameterized in order to be able to determine the algorithm's stability, adaptability, and applicability to occurrences of extreme deviations. The parameter essentially varied the amplitude of the curve which, given the same frequency, results in a more "extreme" curve behavior as the amplitudes are increased subsequently. Some results for this can be seen in Figure 20, Figure 21, and Figure 22. The source code is listed in Appendix B.

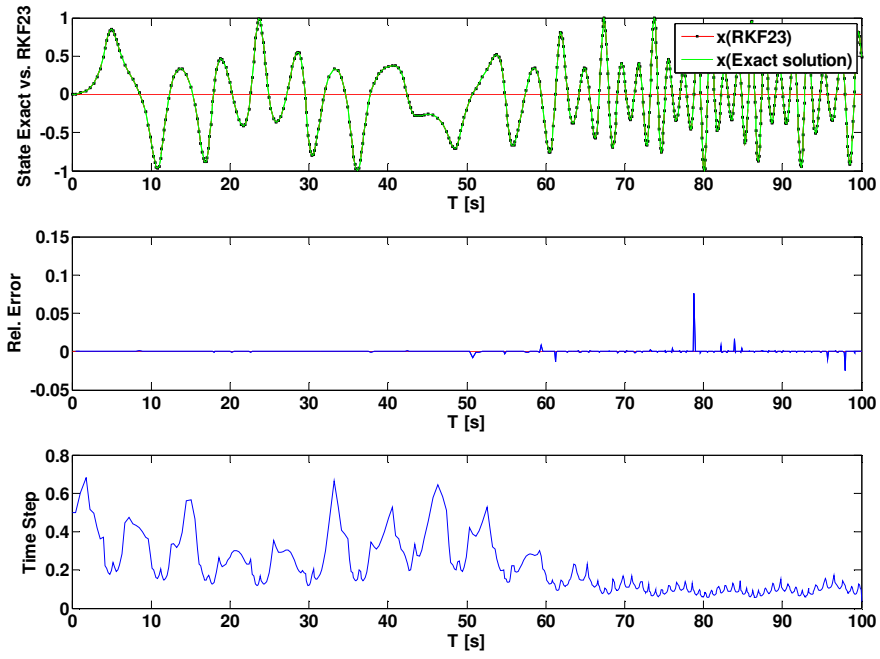

Figure 20. Sample curve with "benign" curve settings (curve amplitude parameter = 2)

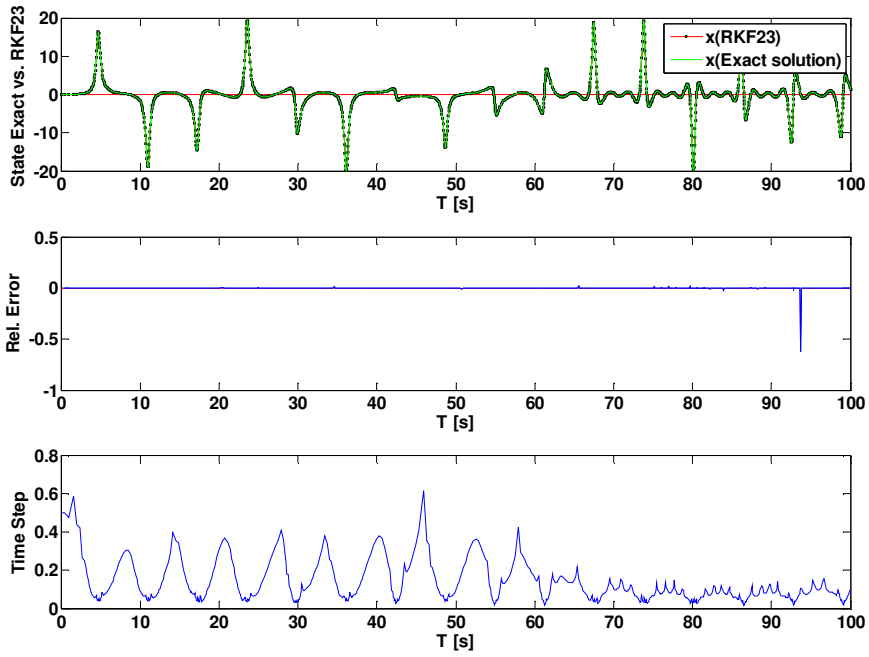

Figure 21. Sample curve with more "aggressive" curve settings (curve amplitude parameter = 1.05)

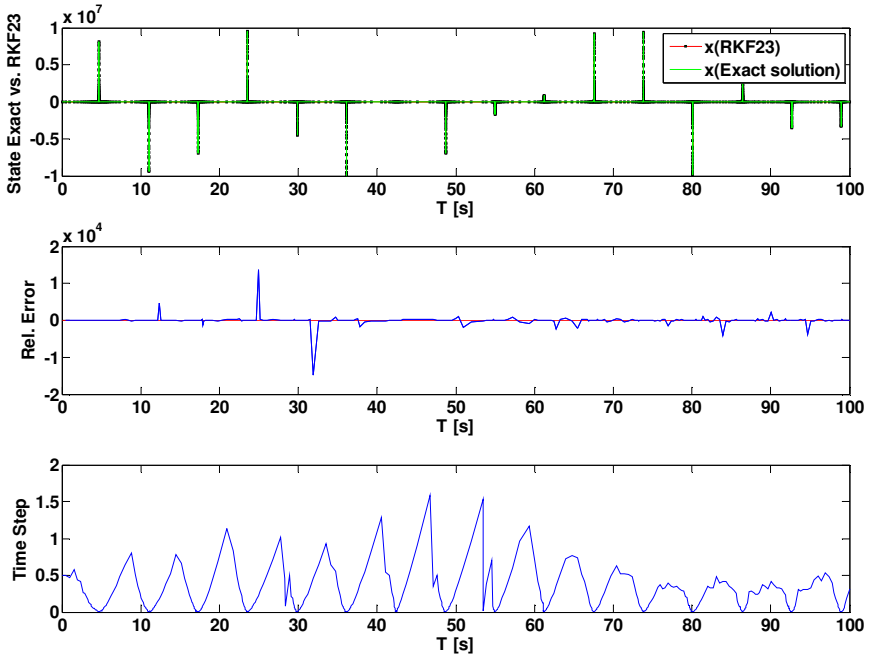

Figure 22. Sample curve with very "aggressive" curve settings (curve amplitude parameter = 1.0000001)

The previous figures show that the RKF23 algorithm is applicable even in extreme conditions. The accuracy is very high. The relative error shows only minor deviations. The deviations are high in cases where the state variable values are close to zero, due to the inherent calculation of relative error. Nevertheless, the agreement between the exact solution and the RKF23 approximation is very good. The graphs also show how the algorithm varies the time step size according to the requirements of smaller time steps = better accuracy at positions where the rate of change of the system state variable is very high. Therefore, the RKF23 algorithm has been proven to work sufficiently accurate for the proposed task.

However, it must be kept in mind that the previous testing of this algorithm was done using a determined function. This is not the case in the final application within a co-simulation. Also, as can be seen in the meta code above, when used with ODEs as underlying equations, the RKF method needs to approximate the solution in order to approximate the overall path of the underlying state variable. An approximate solution, proposed by an algorithm, is not necessary in an actual simulation, since the simulation results are assumed to be accurate. Lastly, the RKF23 algorithm needs the slopes at the points it uses as predictors, correctors, and basis points. In a “Black Box” co-simulation, these points are not likely to be available to the algorithm. The question then is: Why is the RKF23 algorithm proposed for use within a co-simulation environment?

Within a simulation or co-simulation environment as presented in Chapter 4, the RKF23 algorithm is used purely in its role to determine the time step for the next simulation step. The approximated solution is not used because the solution for the next time step is provided by the simulation itself. It does not need to be approximated because it is assumed that the simulation will give the correct output. In other words, the application of the RKF23 algorithm for determination of the next point is not

implemented here. The algorithm is purely used to have a means of determining the time step.

In a “Black Box” simulation, it is assumed that the slopes of the state variables are not given by the models or sub-models. As a relaxation to this constraint, it can be assumed at first that the slopes are given as output parameters of the models or sub-models. But the final application should be the “Black Box” approach without the slopes as given data. In such a case, the RKF23 algorithm can not be implemented directly. Rather, it will be necessary to determine the slopes at the required states and time steps by other means. This will be presented in Chapter 2.

The reason why the RKF23 algorithm is proposed to be used in a simulation or co-simulation environment is that it represents a mathematically rigorous and proven approach towards the time stepping problem. Its derivation and use have undergone proofs on multiple levels, and the algorithm has proven its applicability in numerous applications. As an example, the Matlab functions ODE23 and ODE23s represent direct implementations of the algorithm in a widely used industrial mathematical application software package. Another reason why the algorithms of the RKF family are desirable for use in simulation or co-simulation is that they are capable of handling stiff problems. This is a feature that is strongly desired especially in co-simulation, where the different sub-models can be of different dynamics and thus make the overall integrated system stiff. As discussed before, in such a case, an adaptive time step is crucial to the correct solution of the problem, and the RKF family of algorithms represents a proven means for time step setting. Figure 23 compares the behavior of fixed step methods (Euler and RK2) and the RKF23 adaptive time step in the application to a stiff ODE (y’ = -100*y + 100*t + 101, y(0) = 0.99). Clearly, the fixed step methods can not cope with this equation and a useful time step, while the RKF algorithm manages to fit the exact solution very well.

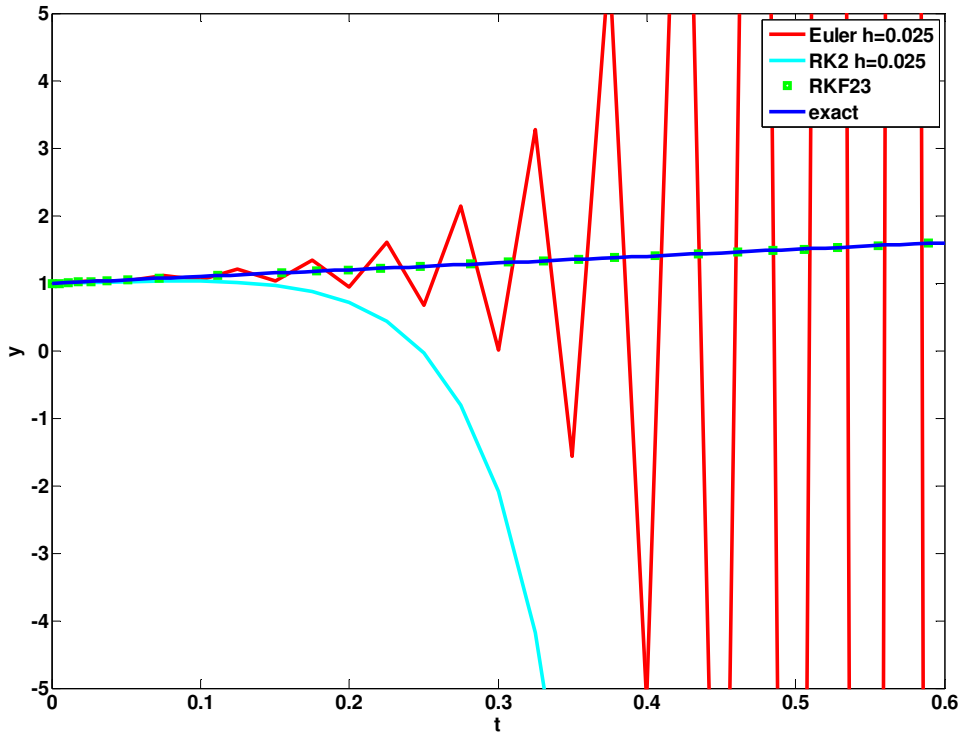

Figure 23. Comparison for numerical solution between fixed and adaptive time step

The RKF23 algorithm has been proposed to be used as the time stepping algorithm of choice, because it also allows for adaptive time step setting and error control but at a cheaper computational expense of only three function calls per simulation point. The algorithm is used only for its ability to set time steps for the simulation to remain within a given error. The approximation part for the state variable within the algorithm is not used. The RKF23 algorithm is a proven and mathematically rigorous algorithm that has proven its worth countless times. It is the proposed method of choice for time step selection within a simulation or co-simulation. This adaptation of theRKF23 algorithm to co-simulation, with all its implications and intricacies, is described in Chapter 4.

CHAPTER 3 CO-SIMULATION OF DYNAMIC MODELS

3.1 Monolithic Models

Currently, most simulation setups consist of a single monolithic system model. Such models are usually mathematical descriptions that represent the essential aspects of an existing real world system which is to be modeled. These mathematical descriptions represent and describe the real world system in such a way that it becomes useable for simulation. In the current text, the models used are assumed to be linear, deterministic, dynamic systems, which indicates that every set of variable states is uniquely determined by parameters in the model and by sets of previous states of these variables, and that the models perform the same way each time they are run with a given set of initial conditions. Further, such models account for the element of time.

In Cellier (1991), a system is characterized by the fact that it can be determined by what belongs to the system, and what does not. Strictly, such a definition leaves room for a "hierarchical" definition of the "system" term: A system can be broken down into sub-pieces, which in turn will also be systems. A system can further be characterized by the fact that it can be determined and specified how the system interacts with its environment. These interactions can be inputs to the system (i.e., they are generated by the environment and fed into the system), and outputs from the system (i.e., they are created by the system and "fed" back into the environment). At least some of the input values should be assignable, and some of its outputs should be observable. Hence, a system can be regarded very generally as a potential source of data. Modeling of such systems is the process of creating an image of a real world system, in order to perform experiments with it and to gather knowledge of the system behavior, and its cause-effect

relations. It allows not only to observe and understand the underlying principles, but also to modify them, and thus not only helps us with analysis, but also with design of such systems.

A model is said to be monolithic if it is modeled in one "block". It has one set of system inputs, and one set of system outputs. It represents the system under investigation as a whole, and does not require any external sub-systems for it to work. It includes all the system differential and algebraic equations, relationships, and data necessary to fully describe the system under investigation. The results from such a model are as accurate as the underlying equations. The equations are being solved simultaneously by a single solver. Such a monolithic model could be represented as a single "box" with only the inputs and outputs showing. An example for such a dynamic system (spring-mass-damper) is given in Figure 24.

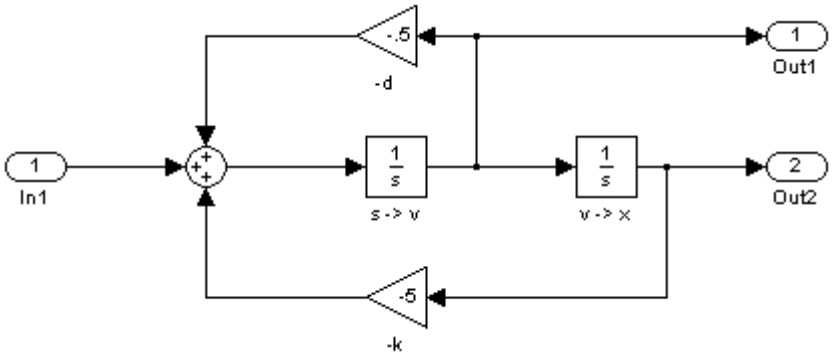

Figure 24. Notional depiction of a monolithic model (modeled in Simulink)

Currently, monolithic models are the prevalent method of modeling systems. Such a model inherently needs to have all the system equations, relations, and data readily known and available, in order for the model to be created. It further requires that the whole model is created with only one and the same modeling tool. All the necessary inputs and outputs must be defined within the model. The implications of these properties are discussed later on in the co-simulation section of this work. For now, it should only be mentioned that a monolithic model uses one and only one solver to solve all the

underlying system equations simultaneously, taking into account all constraints, necessary data, etc. Further, the model has access to computer memory that is shared and fully accessible for all components of the model at any time. These are important implications for the later discussion on distributed models. Figure 25 depicts a notional block representation of a monolithic model.

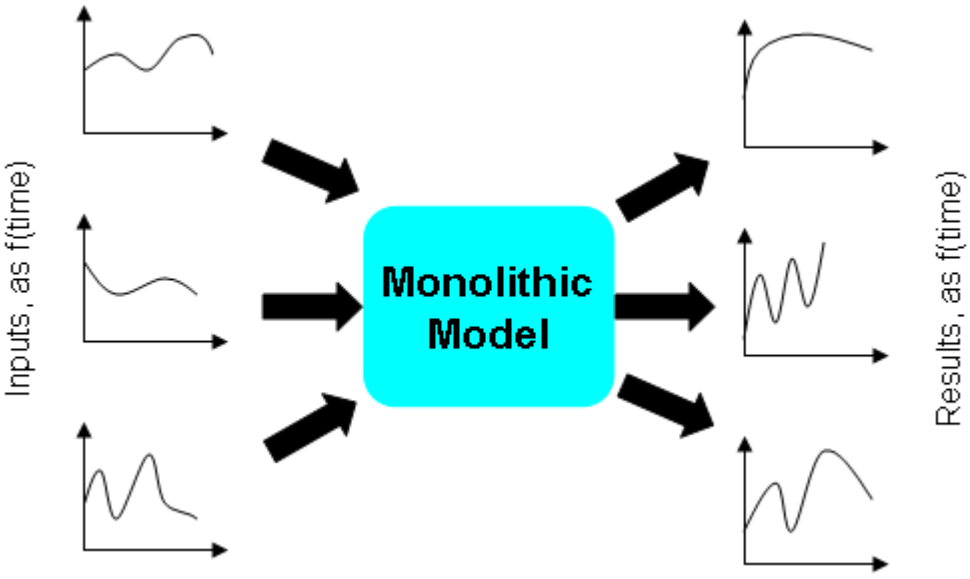

Figure 25. Notional block representation of monolithic model

The following list sums up the properties of a monolithic model:

- One single model within the computer
- Needs all equations, data, parameters, etc. to be known
- All equations, relations, data etc. are known and combined together in one model
- All equations are solved simultaneously by one solver
- Requires whole model to be set up with one specific modeling tool
- Not modular, not easily extendable
- Disciplines bring own terms -> need a "super user" who understands it all and brings it into the model

- Model gets inputs and delivers outputs, but has no interactions and data exchange with other models and no feedback, “one time run”

The last point is of some importance to understand the limits that a purely monolithic modeling approach poses. The monolithic model is executed by providing all inputs required for the model to run. Then, the model is executed for a certain simulation time. The different time notions are discussed later in this text. After this, the model stops execution and delivers its results. If several such monolithic models are employed to solve a particular engineering problem, then they would all be run in isolation. No one model would deliver its outputs to another model. Therefore, interaction effects between models are not caught. Figure 26 depicts the setup of such a configuration.

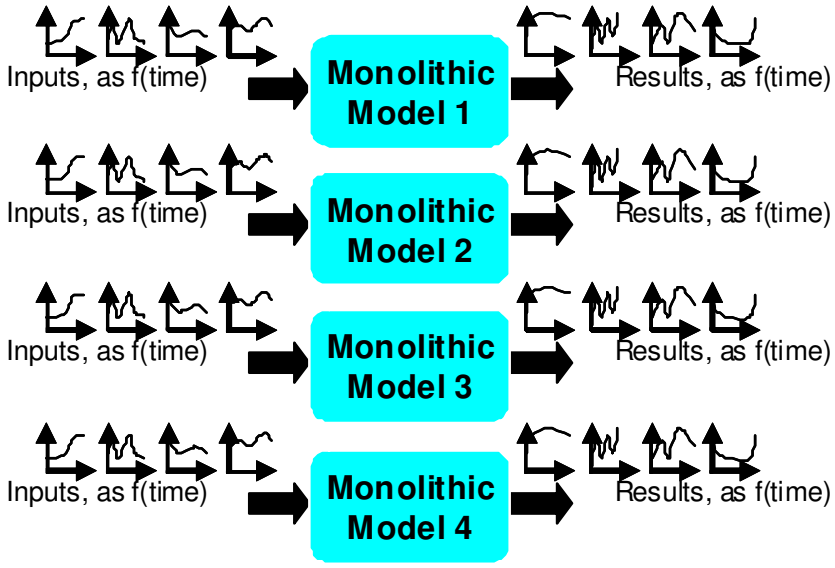

Figure 26. Monolithic models executed in parallel, no interactions, no feedbacks

Monolithic models provide a first approach towards the simulation of engineering systems. However, the modeling of such a monolithic system is oftentimes difficult, if not impossible. The reasons for this are in the inherent requirements described above, and the specific properties such a model must have. A brief list summarizes the aspects of monolithic modeling:

- Some parts of the model may contain proprietary 3rd party information

- Subject matter expert systems and inputs can not be taken into account
- Existing legacy models may have been modeled using different modeling tools and/or languages, and difficult or impossible to be remodeled
- Complex models may require too much computational power
- May require "super user" expert of the model to make modifications
- Assumes that the outputs have no immediate feedback to the inputs (model runs one time, then results are obtained)
- Modeling tools fixed, does not allow for choice of best modeling tool
- Models do not communicate
- Does not allow for easy model maintenance, especially as model size grows
- Does not allow time step de-coupling
- Models built with legacy codes may not be accessible any more
- Models are not shared and not reused, modules can not be easily exchanged
- Does not allow distributed simulation
- Communications among stakeholder of M&S, such as sub-domain experts, simulator users, simulator engineers, and final system users, is not fostered
- Time step interpolation helps to reduce numerical oscillations that may occur if interdependence of coupled variables is very high
- In general, processes, tools, and platforms are not integrated

Monolithic dynamic models face various issues in their modeling and simulation. In general, it must be ensured that the model is well-behaved, and that solutions exist for all possible situations. State singularities may be an example for a case where this is not given. The models may turn out to be unstable under certain conditions and in certain operating states. The internal members or sub-parts within the model may require convergence between their states which may not always be achievable. Further, there

may be constraints between the sub-parts that must be taken into account when solving the model during simulation. Stiff systems must be taken into account, and treated accordingly. It has been mentioned that, for the numerical solution of dynamic systems the time step is of crucial importance for the correct execution of a simulation. The time step directly plays into the accuracy and stability of the model, and will affect almost all the aspects and issues with monolithic models that have been previously mentioned. These issues with monolithic dynamic models are discussed in Chapter 2, with an emphasis on the time step issue and its implications on the simulation of dynamic systems.

As touched on earlier, the underlying system equations and mathematical relations may not be available in full. This might be due to the fact that the system is proprietary. Further, it is very difficult to model a system if diverse disciplines are involved, and their multiple inputs need to be taken into consideration when setting up the model. Oftentimes, subject matter expert opinions from different disciplines will need to be taken into account, but due to the different natures of such inputs it may be difficult or impossible to model the information with only one given modeling tool. Further, such experts may already possess the necessary models, but those are written in different modeling languages, or set up with different modeling and simulation tools. Hence, the existing models cannot be integrated into one large model easily. Oftentimes, these different models are written in legacy codes that are not known any more, cannot easily be modified by the current users, or are not accessible due to the proprietary nature of the model. Lastly, monolithic models will require large computational resources, since they represent oftentimes large and complex systems. Since they are modeled as one block that can not be split or separated in any way, multi-processing capabilities that have evolved and been optimized in recent years can not be utilized to increase the computational efficiency of such models. For all the above reasons, monolithic models are oftentimes not employable easily, or it is altogether impossible to set them up.

However, as discussed earlier, more and more often complex systems need to be modeled by combining different sub-models from different partakers and collaborators of a project. These models are often proprietary, and their underlying equations, mathematical relations, and system behavior is not known. In fact, the only information available for such models are the input state variables the models requires being able to execute its intended purpose, and the output state variables that the model delivers to fulfill its purpose within the simulation. It follows that such systems will need to be brought and linked together in a way that makes them work together, exchange their data, and represent the overall system in and efficient and accurate manner.

3.2 Co-Simulation

The technique of linking third party models together and simulating the resulting overall system in such a manner is commonly referred to as co-simulation. Several tools exist for this task, such as Phoenix Integration's ModelCenter, Engineous Software's iSight, TLK-Thermo GmbH's TLK Inter Software Connector (TISC), and others. They are co-simulation environments for controlling different simulation applications and exchanging data between them. They are used to organize the co-simulation setup by managing simulation programs, models, parameter and initialization settings, different simulation computers as well as the IP connections. During a co-simulation, they exchange the data between the simulation programs and models, synchronize data, handle events and oftentimes have the capability to graphically report the exchanged data. They have interfaces to be able to easily integrate and process the most common modeling and simulation software tools into the integration environment, such as Flowmaster, Fluent, LabVIEW, MATLAB/Simulink, Modelica/Dymola, Modelica/SimulationX, STAR-CD, THESEUS-FE, WAVE, and others. It can be seen that the useable modeling and simulation tools that can be integrated include a wide variety of applications, such as

general modeling and simulation (Matlab/Simulink), modeling of dynamic systems (Modelica/Dymola), CFD (computational fluid dynamics, e.g. STAR-CD and Fluent), pipe flows (Flowmaster), engineering system models (Labview), and others. Particular applications also include multi-body dynamics (for example, astrophysics and orbital mechanics), electrical circuit design and analysis, molecular dynamics, computer networks, structural dynamics, finite element methods (FEM), CFD and fluid flows, thermal and heat flows, power systems, crack growth and creep analysis, and many more. Oftentimes, such integration environments also provide comprehensive programming languages, a feature that will be discussed later in the context of co-simulation optimization.

Figure 27 depicts how a single monolithic model is transformed into a co-simulation setup. Figure 28 depicts the notional setup of a co-simulation.

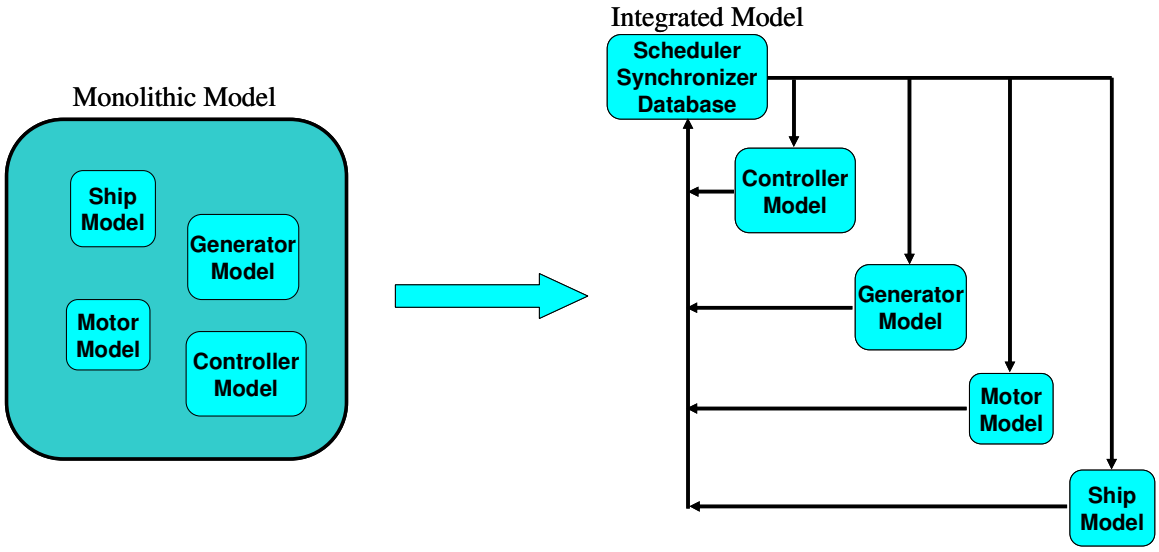

Figure 27. Transformation of a monolithic system into a co-simulation

The models are linked together not directly, but via a scheduler that fulfills a critical role in the linking of the sub-models by fulfilling the following tasks:

- Data input and output synchronization
- Data assignment

- Time step setting
- Data modification, as needed (e.g., unit conversions between models)
- Data storage in database
- Model execution according to schedule

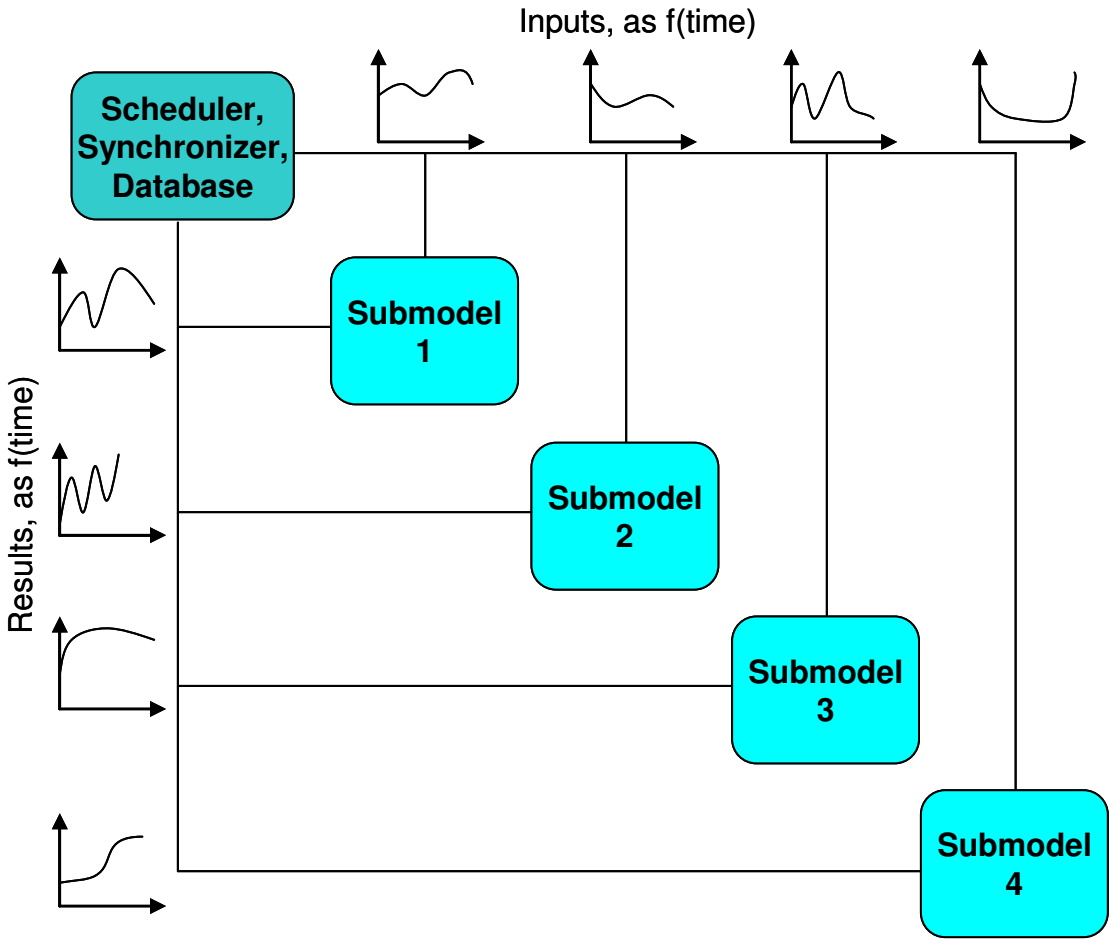

Figure 28. Notional co-simulation setup

The sub-models that we need to integrate are assumed to be dynamic engineering systems. They are assumed to be:

- Heterogeneous
- Multi-scale (in time and space, e.g. different dynamic behaviors such as “fast” vs. “slow” dynamics)

- Highly coupled; change in one system affects all others and overall system
- Complex by themselves
- Decoupled in a sense that there exist no algebraic loop between them (Kubler and Schiehlen (2000) describe a co-simulation application whose extended and improved setup considers such loops between sub-models. For the context of this thesis however, the invoking of such loops into the simulation would only complicate the problem, but not provide any advantage in the development of the methods. Therefore, the assumption will be that such loops do not exist).
- Well-behaved, stable, and free of algebraic loops and singularities, such as infinite jumps, divisions by zero, etc.; such problems could to a certain extent be taken care of within a co-simulation, but are assumed to be non-existing for this text. The models are also assumed to be free of blow ups, as described e.g. in Acosta et al. (2002)
- "Black Boxes"; their "internals" (= their mathematical equations and descriptions) are unknown, hence their dynamic behaviors are also unknown. This situation can arise even if the modeling engineer is the one to model the system: Despite the fact that the build-up of the model is known, the system equations still remain unknown. This can be the case if a modeling environment is used that allows the creation of models without explicitly stating the underlying equations
- Giving only state variables as outputs, and expecting only state variables as inputs. No derivatives or miscellaneous outputs are provided to help identifying the dynamic system behavior
- Continuous
- Linear (the implications on this are discussed later)
- Time-invariant
- Smooth; the Lipschitz criterion is fulfilled everywhere

- Dynamic, time-dependent

Time-invariance is different than time dependent, as listed above. Time dependence means that as simulation time progresses during the simulation, the dynamic system will behave in a certain way and the state variable will change according to the underlying model. Time invariance means that the system will always behave the same way if started with the same initial conditions and parameters, no matter at what real world "wall clock" time and, more importantly, no matter at what simulation time.

The Lipschitz condition that is required to be fulfilled (the Lipschitz continuity, named after Rudolph Lipschitz, a German mathematician) can be understood as a form of uniform continuity for functions. It is related to the slope of a function at a point. It states basically that for every point along the simulation states, a variable has to have a slope that is lower than or equal to a certain constant, the Lipschitz constant. It guarantees the existence and uniqueness of the solution to an initial value problem for a differential equation, where the slope is a vital metric for the numerical solution of such a function. Since the slope of a function will be a critical component of the algorithm discussed further below, this condition is critical to the practical applicability of the proposed algorithm. It must be noted here that in the numerical integration of ordinary differential equations with digital computers, as described below for the proposed time stepping algorithm, the Lipschitz condition is always fulfilled unless a state variable become infinitely large (either positive or negative). Infinitely large state variable would result in infinite jumps. Unless this is the case, the slope of a state variable from a model will always be defined because due to the digital nature of computers, a finite time step will need to be introduced to numerically solve the underlying equations. Since the time step is always finite, and the state variable value is assumed to be finite everywhere as well, the slope will be defined at all points, despite the fact that it may become excessively large.

The last point in the previous list, “dynamic” system, is critical to the path this text will go along. Since the solutions discussed in this text are derived from the solutions of time-dependent systems and algorithms, the proposed solutions can only work if the underlying systems are also of this type. In particular, this means that their behavior must be time dependent. Time will be the independent variable during the execution. The time step selection will be the critical metric for the accurate and efficient execution of a co-simulation. Time step selection will be discussed in the respective chapters. Inherently, a sub-model can not be discrete, because there will be no “slope” metric for discrete states.

The two points “Black Box” and “state variable input and output” are also critical in the definition of the problem. Many instances are known when all sorts of intelligent algorithms have been used to solve the equations of dynamics systems, or to co-simulate various different models. In all those instances however, the underlying equations were known and accessible to the solver. In this case, the numerical treatment of the problem is straightforward, and conventional algorithms can easily be applied. In a co-simulation where the sub-models are black boxes however, such numerical treatment is not possible since the underlying equations are not known. The co-simulation only has input and output states to work with, and to use for derivation and inference of system behaviors. Based on this information, the co-simulation must decide how to set the time step and treat the overall system model. This is even more complicated since common algorithms will not be directly applicable since a co-simulation does not behave like an equation. This topic is discussed further below.

The “Black Box” assumption can be relaxed to a certain degree if “insights” into the underlying model can be gained, for example through system identification techniques. System identification is a special field in controls, and allows for the evaluation and estimation of equations of an underlying system. It has a severe limitation, namely that it’s technique is only applicable to linear systems. Linear systems are systems that essentially will, under all operation conditions, double their outputs if their

inputs are doubled. The output-input relation is linear. If the linearity assumption is not given, common system identification methods will fail. It is difficult to impossible to actually confirm system linearity over all possible operating conditions of the simulation. Thus, system linearity must be either assumed or known to be fulfilled, if system identification methods are to be employed. Another possible approach towards understanding the system internals and possibly simulating them as a surrogate models is the more current method of modeling such systems through a neural network. Such neural network are being trained by applying known and controlled inputs into the underlying unknown model and observe the resulting outputs. These outputs then are used to train a neural network which will represent the underlying model in a similar manner, and can be used as a surrogate model. However, from statistics (a field in which neural networks are used very often for non-linear regression analysis) it is known that the parameters of the neural network model are not easy to interpret as model parameters. These parameters can not easily be interpreted e.g. as a damping ratio or spring constant of a dynamic model. They thus give only limited insight towards the actual system equations, and therefore represent only a limited usefulness for the problem of co-simulation of "Black Boxes".

3.3 Issues With Co-Simulation

When linking third-party models together through a co-simulation, there are several issues to be considered. Such issues include, but are not limited to, the following:

- sub-model stability (both physical and model stability)
- linking of sub-models that need convergence for each time step
- existence and uniqueness of solutions
- smoothness of function everywhere
- transients vs. steady states

- general well-behavedness of sub-models
- singularities within the sub-models (e.g., infinite jumps, divisions by zero)
- non-linear interactions between sub-models
- constraints between the sub-models
- algebraic or feedback loops
- parallel computers
- variable exchange
- surrogate models, system identification
- coupling strong vs. loose; how does information have to be passed in such conditions
- system timings (some systems might not have to be run unless certain conditions are given)
- sub-model observability
- etc.

Previously, it was discussed that if a dynamic system model is to be simulated in a digital computer, actual continuous time simulation is not possible. Rather, the simulation time must be split into discrete time steps at which the simulation model is evaluated and the states exchanged and stored. This time discretization introduces certain issues, but can also be used to impact on the accuracy and stability of the simulation. Such issues with particular focus on, and cause due to, time stepping are, among others:

- stiff systems; bypassing of slow systems
- accuracy of the integrated overall system
- sudden system state changes
- changes in model dynamics during simulation execution

- linking of dynamic and discrete time sub-models

This list of time stepping issues is not exhaustive, but it does present some of the most important issues to be taken care of in continuous time simulation on a digital computer. This thesis text will mainly talk about these issues, how they relate to co-simulation, and how the time step setting can help circumvent simulation problems.

To give a quick look at how the time step affects the error of a simulation, a simple simulation was run with a successively increasing time step. The base case is the simulation run with a very small time step. Subsequent runs were compared to this base case run. The underlying model was a simple pendulum model whose frequency is known (for the linear case of small angles, which was the case here). For every comparison case, the deviation from the base case was calculated using a root mean square error (RMSE). To have equal comparisons, the simulation was set up such that the pendulum made 10 complete swings. Several such comparison runs were performed for different system dynamics by changing the length of the pendulum bar. Through this approach, a notional graph could be created that shows the root mean square error as a function of the system frequency and the time step chosen.

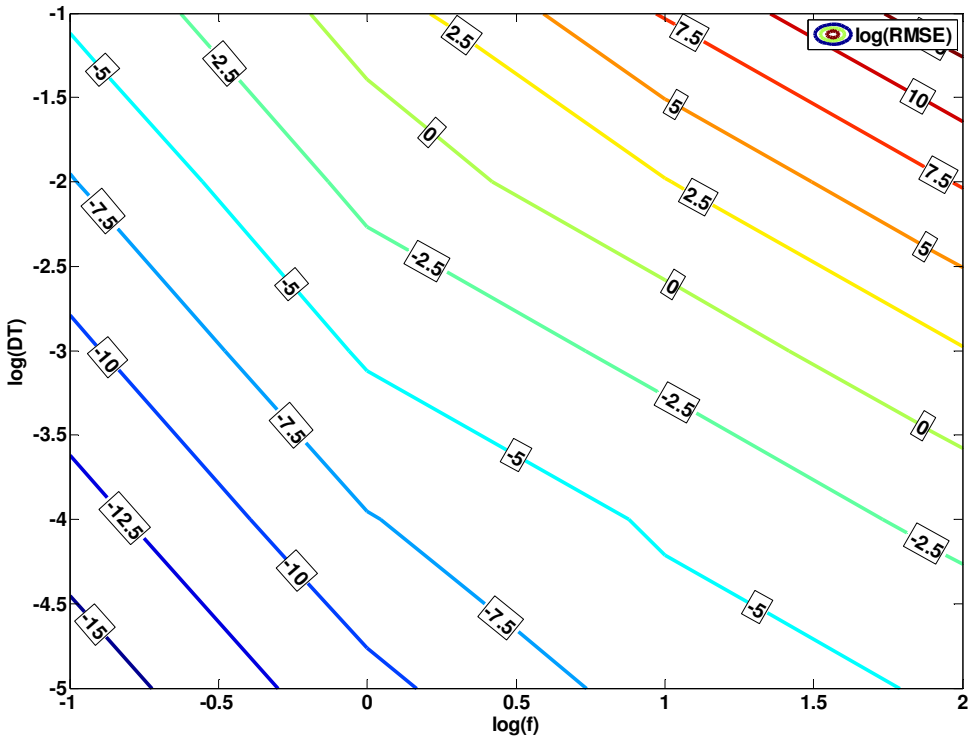

Figure 29. Notional graph of root mean square error (RMSE) as f(system eigenfrequency, time step). Note triple logarithmic scale

It is clear from this graph, that the effect of time stepping on the deviation from the base case, and hence the error, is significant. Increased time steps reduce the simulation accuracy. The error depends on the time step and the dynamics of the model. Faster dynamics will require smaller time steps in order to keep the error within certain limits. The model in this example had constant dynamic behavior. If this is not the case (like in most dynamic system simulations), and the system dynamics change during the simulation execution, the time step must be adapted to the changing system behavior. This graph is only notional, as it will be shown that an adaptive time step would change the error in ways specific to the time stepping algorithm used. This text will further elaborate on such scenarios, and propose a time stepping algorithm for co-simulation setups.

3.3.1 Data exchange synchronization

First, it must be determined which variables need to be exchanged between the sub-systems. This is important, because it determines not only the computational effort that the co-simulation environment has to go through in order to execute the models correctly and exchange the correct data between them. It is also important because it may be a limiting factor for a co-simulation, namely when variables required by some sub-models are not actually provided by others. In the special case of dynamic sub-systems that are represented by their state space equations, a first derivative value may need to be provided for such models, however, this derivative may not be available for the simulation. Methods for evaluating such a derivative during the simulation may need to be applied and tested. In general, during the execution of one time step in a co-simulation, It is assumed that the outputs y have no feedback to the inputs u. Only after a time step has been executed and the models have stopped, the variables will be fed into the scheduler and redistributed accordingly. The issue of data exchange is especially critical when discrete models are part of the integrated model. Discrete models, as discussed before, do not change continuously over time, but change states only at certain times in a discrete fashion. When using continuous time simulation, the time step of the integrated model must be set with the discrete model in mind. It must be ensured that any change of the discrete model's states will be taken into account when it happens, and not with some time delay due to the time stepping scheme one must apply in continuous time simulation on digital computers.

3.3.2 Coupling Between Models

One must also consider the way the sub-models are coupled. Basically, there are two different methods. One is commonly referred to quasi-dynamic coupling, loose

coupling, or ping-pong coupling (Zhai (2004), Struler et al. (2000), Hensen (1999)). Here, distributed models run in sequence, and one model uses the known output values, based on the values at the previous time steps, of the coupled model. The feedback between the programs is lagged one coupling time step. The other main coupling method is usually named fully dynamic coupling, strong coupling, or more colloquial, "onion coupling". In this method, distributed models iterate within each time step until the error estimate falls within a predefined tolerance. The coupling used in this context is more based on the first method. The models are run in "parallel" (this is of course not possible in a strict sense, at least not on a one-processor computer. "Parallel" in this case means that all models are run one by one, without any exchange of variables, and each model waits its turn and after execution, waits until all other models have also finished their current run), and the state variables are fed into the scheduler and re-distributed after all sub-models have finished their turn. The application of such coupling strategies in a co-simulation are described in Trckal (2007). This paper also hints on the use of adaptive time stepping within a co-simulation, but does not elaborate further how these time steps were determined. A time stepping algorithm is used within one of the sub-models' execution tools. This would indicate that the time stepping is only a "local" time step (as defined later in this thesis). Hence, the model description equations would have to be known, and common time-stepping algorithms for solving such equations can be employed, such as RKF23 (described below). Another method that has been presented is a "gluing" algorithm. Wang et al. (2003) demonstrate their approach which relies only on information available at the subsystem interface levels. Hence, models can be analyzed at their distributed locations, using their own independent solvers, and on their own platforms. They further elaborate on their approach in Wang et al. (2005). Tomulik and Fraczek (2010) demonstrate the applicability of the gluing algorithm on a double pendulum example.

3.3.3 Constraints Between Models

The second major issue may arise due to constraints between the sub-models. A constraint exists if some state of a sub-model directly impacts some state of another sub-model. Consider a refrigeration cycle as illustrated in Figure 30 with a causal conflict existing in input-output relationships (Gu et al., 2004). The system consists of a compressor, a condenser, two sets of a series combination of evaporator and expansion valve, and an accumulator, along with pipes connecting those components. Refrigerant is first pressurized at the compressor, liquidized in the condenser, and branched out to the two indoor units installed in different rooms. The outlets of the evaporators are merged, and the refrigerant is collected at the accumulator. Interactions occur at this merging point, where mass flow rates from the two evaporators, u_A , u_B , sum to the mass flow rate in the pipe reaching the accumulator:

$$u_A + u_B = \dot{m}$$

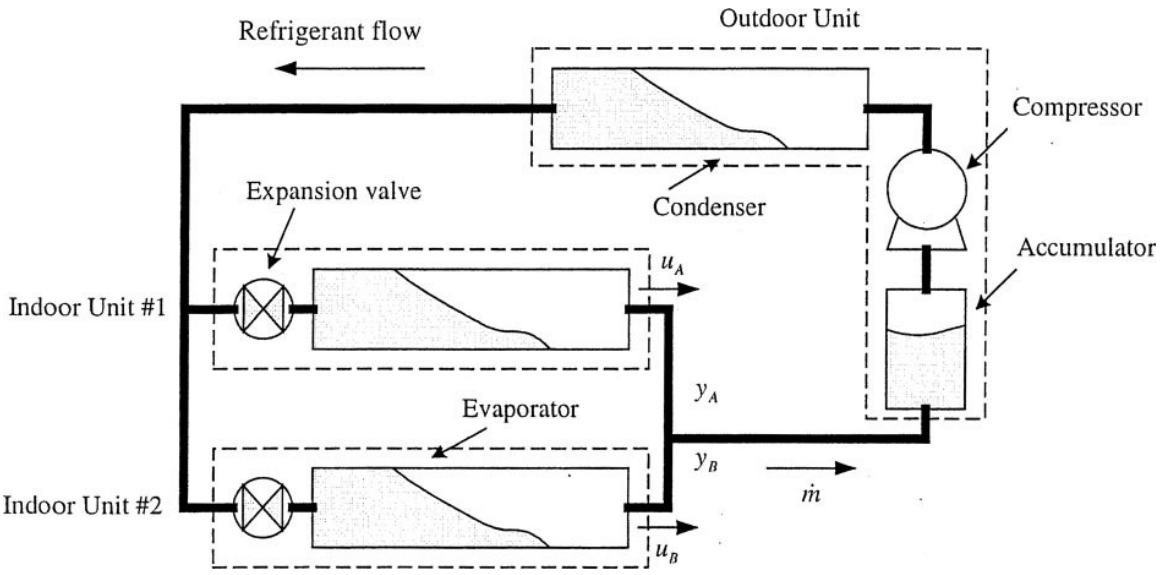

Figure 30. Causal conflict in refrigeration cycle model

Let z be an independent variable, called a boundary variable, introduced to rewrite the above condition as:

$$u_A = z, \quad u_B = \dot{m} - z$$

When the two evaporators are placed in adjacent rooms connected by a short pipe, there is no tangible pressure difference between the two evaporator's outlets, denoted y_A and y_B. Namely, $y_A = y_B$. In describing the dynamics of each evaporator the outlet mass flow rate appears as a subset of the inputs and the outlet pressure appears as a subset of the outputs, and their input-output relationship cannot be reversed (Gordon, 2000). Therefore, a conflict occurs when combining the two evaporators, as shown in the figure. Both subsystems provide outputs that must be the same, while the inputs must conform to the form that introduced the boundary variable. This type of problem is often encountered when combining multiple subsystems. Subsystem inputs and outputs must conform to certain algebraic constraints; hence the total coupled system is described by Differential Algebraic Equations (DAEs). Two robot arms connected at the endpoints and a four-bar-linkage system are classical examples of DAE. If such constraints exist, and hence the system must be modeled using DAEs, it will complicate the co-simulation substantially, since the constraints will need to be taken into account when exchanging data between the sub-models. In such a case, state variables may not be sufficient for the co-simulation to work properly, and algebraic constraints will need to be modeled into the co-simulation. For the case when the equations are known, Wang et al. (2001) present an approach using implicit Runge-Kutta methods with adaptive step time integration, and apply it to a specific example (transient magnetic fields). Similarly, Emel'yanenko (2007) uses an adaptive time step for symplectic integrations for the solution of the planetary N-body problem, for which the equations are known. However, in the remainder of this text, the sub-models are assumed to be without causal conflicts as described above. This is a safe assumption because the causal conflicts themselves do not change the approach

towards the model developed in this text, but would unnecessarily complicate the setup and testing of sample models, in addition to making the process more prone to error. A special case for the solution of DAEs is presented in Cameron at al. (1998). The authors use an eddy current model for their investigations, and present two Runge-Kutta methods that are suitable for the time integration of the classes of DAEs to which eddy current problems belong. Both their proposed methods have error estimators and hence allow variable step sizes. In their tests, the variable step size integrators were competitive with fixed step size integrators, in particular with Crank-Nicolson (also known as the Euler second-order trapezoidal rule (Hairer 1987), suitable for the solution of PDEs). Their specific approach is explained by the fact that for transient eddy current problems that are modeled as differential-algebraic equations (DAEs), a time integration method suitable for ordinary differential equations (ODEs) might not necessarily work.

3.3.4 Simulation Stability

In dynamic systems, the term “Stability” expresses the development of the solution over a period of time when the initial conditions are perturbed by a small amount. If a small perturbation of the initial conditions results only in a small perturbation of the results, then the system is considered stable. For stability in the context of simulation and co-simulation, it must be considered whether “system” (or “physical”) stability or “model” stability is the issue. “System” stability refers to the underlying dynamic system, and is not related to modeling and simulation. The underlying dynamic system can be unstable in the sense described above. In such a case, a modeling and simulation approach will be very hard, because it will be difficult to capture the instabilities with the simulation. If the dynamic system is modeled and simulated in a computer, then it can become unstable during the simulation even though the underlying dynamic system is by itself physically stable. This instability is the “model” instability, and itself strongly dependent on the time step chosen for the

simulation execution. The “system” stability was assumed to be given in the model assumptions stated earlier.

In the stepwise numerical solution of differential equations, as it is necessary when running simulations on digital computers, the end result of one time step is the starting point (initial condition) of the following time step. If the perturbation of the initial condition of a time step has an impact on the result of that time step, then these impacts and perturbations will continue to evolve through the simulation run. At every time step, a new perturbation of the initial conditions will add yet more perturbation to the results of this time step. If not properly formulated and simulated, such simulation setups will result in unstable simulation results. In general, such results will show in erratic behavior of the simulation output. This is especially critical in co-simulation where multiple input states receive data from different models. Each of those inputs will have small errors, which will then perturb through the respective sub-model and handed on to the next sub-model during the simulation runs. Such behavior is of course not acceptable. In continuous time simulation, the stability is mostly determined by the simulation time step. Too large a time step will in almost any case result in unstable simulations and unacceptable results. Figure 31 depicts a sample output of a simulation (the model is a single bar of a double pendulum co-simulation setup, which will be described in great detail in Chapter 4).

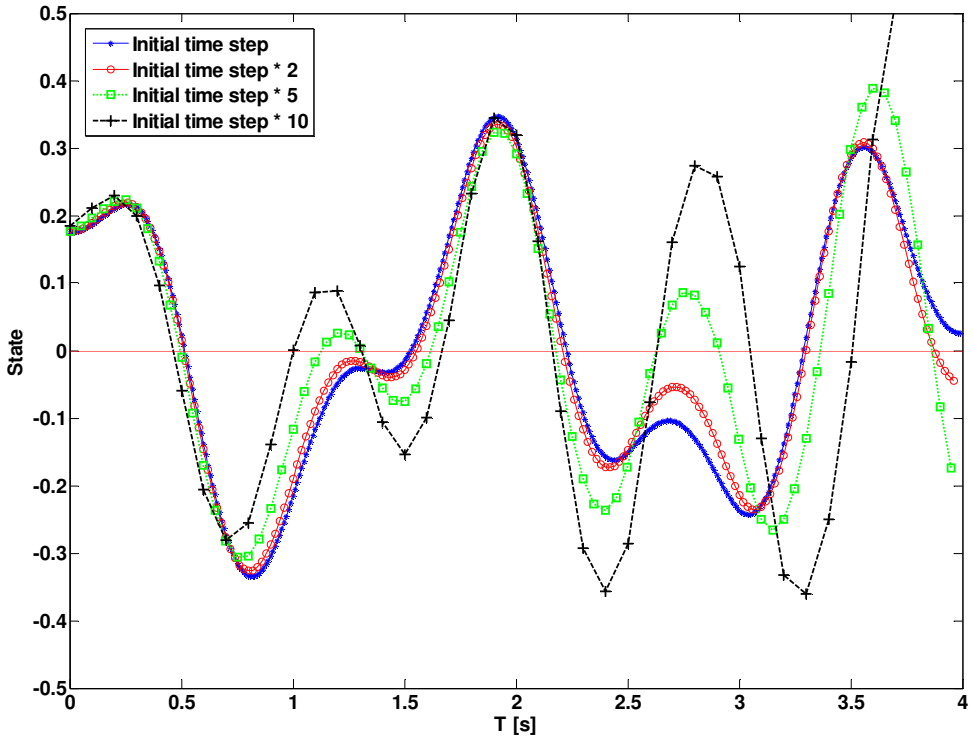

Figure 31. Instability as a result of time steps chosen too large

With a sufficiently small time step, the simulation is well-behaved and smooth. As the time step is increased, the results start to deviate from the actual solution. This effect is weak with small time step increases, but becomes very strong as the time step is further increased. As time step and simulation time increase, the results do not represent the actual system correctly any more. In this particular example, the solution for the "Initial time step * 10" case actually runs out of bounds towards the end of the simulation run. This is then where the model becomes unstable (which does not mean that the underlying physical system is actually unstable, but only that the simulation is not accurate but rather, unstable).

3.3.5 Singularities

When simulating continuous time dynamic models, a situation can occur where the model reaches a singularity. This may be due to illegal mathematical operations (e.g., a division by zero), due to non-convergence because of number resolution issues, due to algebraic loops, etc. For single models, there might be a time constraint invoked that stops the simulation when a certain predetermined time is exceeded. For co-simulation, the setup must make sure that it can handle the case when a sub-model has reached a singular state. This will mean that the sub-model will not deliver results any more while still giving the impression as if it is still working on its current run. Also, a co-simulation can itself enter a singular state, since the integrated model itself is a dynamic model. The opportunities are more widespread in the co-simulation case, because cases that are valid for single dynamic models apply here as well as other issues, such as inter-model convergence and constraints as described above. The co-simulation scheduler script will need to make sure that for example convergences will not result in infinite loops, that algebraic loops are avoided, that constraints are not violated, etc.

3.3.6 Stiff Systems

In monolithic dynamic systems, the definition of "stiff" systems is somewhat debated. This is due to the nature of differential equations, which count as stiff if the solution with numerical methods results in instabilities unless the time step chosen is extremely small. Here, the terms "unstable" and "small" are somewhat arbitrary and depend on the underlying system. Therefore, stiffness is not a generally applicable metric. Stiff systems will result in highly inaccurate and possibly oscillating solutions when using fixed time step methods such as Euler of Heun's method (see above). In order to solve such systems accurately, adaptive time stepping methods should be applied. For co-simulation, the stiffness criterion must be taken into account for the sub-models, and also can be extended to the different dynamic behaviors that the sub-models will show. When connecting different sub-models together to form an overall integrated dynamic

system, the different dynamic behaviors and properties of the sub-models make the integrated model a stiff system and require a distinct treatment of each sub-model to ensure that the integrated simulation does not oscillate or become inaccurate. This requires the careful selection of applicable and adaptive time steps for each sub-model, both on the sub-model and the integrated model level. While in most co-simulation applications the sub-model level time step is set by the respective solvers of each sub-model, the overall time step with which the data is exchanged between the sub-models (which represents the solution time step) must be chosen by the co-simulation scheduler script in a fashion that ensures the proper execution of the stiff system.

3.3.7 Hardware-In-The-Loop Systems

Hardware-in-the-Loop (HIL) systems present a specific challenge to co-simulation. HIL means that some parts of the integrated simulation setup are simulated in a computer, but this computer is connected to real world hardware which acts as one or multiple sub-models in the overall simulation. A simple example would be a human driver in a driving simulator. The steering wheel will give real time inputs into the simulator. Such setups present specific challenges because they enforce real time signal processing and data exchange between the sub-models. This becomes even more critical when one looks at the numerical methods that are often used to solve dynamic models in a computer, as introduced above. It has been shown that as the time step reduces, the error of the simulation also reduces, $\varepsilon \rightarrow 0$ as $h \rightarrow 0$. Hardware has an infinitely small time step, and hence the error that the hardware delivers is zero. The computer simulation must take into consideration the infinitely small time step when determining how to execute its computer models with respect to the hardware inputs. This is also a problem since the computer can read out the hardware only with finite accuracy due to signal data resolution and data sampling frequency. A co-simulation setup with HIL components thus requires the programming of real time applications that require the allocation of the

full computer capacities available with the consideration of all the issues described here. Furthermore, and with respect to the subject of this thesis, a HIL simulation usually has the problem that it can not access state derivatives directly from the hardware (it has been shown before that for the solution of ODEs, such as those used to describe dynamic systems, the state derivative is necessary). Also, in many cases the equations to describe the HIL will not be available, or merely approximations of the real underlying system behavior descriptions (for example, if certain properties of the hardware are unknown). Hence, HIL simulation is a very special case of co-simulation, and will not be treated in this text.

The previous points listed here represent a cross section of the problems and intricacies that combining different sub-models to an integrated simulation will arise. While some of those problems are inherent to the formulation and correct running of the sub-models, it has also been shown that one critically important metric for a dynamic system co-simulation is the time step with which the models are executed and run. Stability can only be ensured when the time step is handled correctly. Stiff systems will only run correctly when the time step of the integrated simulation is set and maintained adaptively and correctly. The overall time step of a co-simulation is not the only criterion that needs to be taken into consideration, as the previous list has shown. However, it is safe to say that all other considerations are futile if the very basic issue of time step setting is not solved and handled correctly. Without a properly set global simulation time step, the simulation results will be inaccurate at best; the simulation will become unstable and crash at worst. It is therefore of critical importance to find ways which help to determine and set the global simulation time step appropriately, in order to enable correct co-simulation and thus the solution of complex design problems. Therefore, and with the intricacies of “Black Box” sub-models in mind, this thesis text will focus on the development of a time stepping algorithm for the integrated simulation of continuous time dynamic models.

This chapter has shown the properties of monolithic dynamic models. It discussed the intricacies of such models, and the necessity of extending the concept of monolithic models and integrating them into an overall dynamic system model through the technique of co-simulation. It has described the properties of co-simulation, shed light on some of the issues that come with co-simulation, and offered an insight into the rationale and vital importance of time stepping of the integrated system. The following chapter will introduce the application of the discussed time stepping methods into the co-simulation environment. It will show the implementation of the RKF23 algorithm to a described toy problem, discuss results, and show extensions of the basic algorithms with respect to specifics of co-simulation.

CHAPTER 4 APPLICATION OF TIME STEPPING ALGORITHM TO CO-SIMULATION OF DYNAMIC MODELS

In the previous chapters, the application of dynamic system models for simulation has been introduced. The intricacies of dynamic systems have been laid out, and the rationale for an adaptive time step for the numerical execution of dynamic system models has been given. The necessity for linking dynamic system models into an overall integrated system model through co-simulation techniques has been explained. The necessity for adaptive time steps in the extended scope of co-simulation has been rationalized. Lastly, the idea behind the application of proven time stepping algorithms to solve the time stepping problem in "Black Box" co-simulation has been introduced and explained. With this background knowledge given, the final step of implementing such a time stepping algorithm to a co-simulation setup can now be made. This chapter will introduce a simple dynamic system that is decomposed into its sub-systems, and use this as a toy problem to demonstrate the development and implementation of time stepping algorithms with particular attention to the properties of co-simulation. Time stepping schemes will be illustrated. The results will be discussed, and extensions will be made to accommodate for the intricacies and specialties that a co-simulation represents compared to a simple monolithic dynamic system model.

As was shown, for dynamic simulations in general the time step is a metric that has utmost importance and criticality to the stability and accuracy of the simulation results. For monolithic models with known equations, there exist mathematical algorithms to solve for an "optimal" time step that keeps the amount of function executions low while retaining prescribed simulation execution accuracy. An integrated simulation setup that consists of multiple dynamic models is itself a dynamic model, and will therefore need to put particular weight on the selection of its time step. The difficulty with co-simulated integrated models is that their mathematical description is oftentimes

not easily accessible or not available at all. Under such circumstances it is not possible to apply strict mathematical algorithms that can determine a time step setting easily. In such conditions, only the model outputs are available as metric based on which a time step can be set or at least estimated. The purpose of this thesis is to find an algorithm that can be used similar to the existing algorithms for monolithic models with known equations and adapt that algorithm to the problem at hand, namely the setting of the global time step ΔT for a co-simulation. The problem is not trivial since the different dynamics of the sub-models must be taken into account. The data exchange and possible complex interactions between the sub-models must be taken into account when setting the co-simulation time step, due to their impact on the simulation output accuracy and stability of the simulation. As it will turn out, there can be more than one setting for ΔT during the co-simulation execution due to the possible different sub-model dynamics, and the time step setting algorithm will need to be extended to take care of such problems. This is the main part of this text.

The second issue is again regarding the time stepping (and hence demonstrates already the importance of the time step on simulation execution time and accuracy), and takes into consideration the amount of real world time used by the simulation versus the accuracy of the simulation results. The selection of the time step size will offer great potential for the improvement of the co-simulation execution speed and accuracy, but it also poses issues regarding the selection of "optimal" time steps, especially with sub-systems whose natural frequencies vary by orders of magnitude. If models with high natural frequencies need to be co-simulated with models with low natural frequencies, the issue will be when to exchange system data between the models. This problem is similar to the problem of numerical integration of stiff dynamic systems. During the course of this text, it will be investigated whether it is possible to "skip" slower sub-models for a certain amount of time steps, and to only run such sub-models that have high system frequencies. Naturally, the event of sudden system changes will need to be taken into

account, and the effects of such sudden changes, and remedies against their impact on the simulation, will be discussed.

The third issue is of paramount importance for the development of a method to determine "optimal" time step is the fact that a co-simulated model does not behave like an equation. While the sub-models are modeled using system equations (which usually are, for dynamic models, ODEs and/or PDEs; the difference is that ODEs usually have possible multiple instances of one variable as a parameter [for example, time], whereas PDEs have a mixture of several such variable parameters to solve for [e.g., time and space]), and hence can be solved using common algorithms (which may include time step variations), the combined sub-models in the co-simulated model do *not* behave like such an equation. In other words, the overall simulation is *not* representable by an equation or set of equations. This means that algorithms for the solution of sub-models can not directly be employed to the solution of a co-simulation model, because the underlying equations are unknown and not exactly derivable from the sub-models. While the claim can be made that such underlying equations could be derived, e.g. by means of system identification and adapted surrogate models, any such inference about the unknown sub-model behaviors introduces additional errors that would further falsify the approach towards "optimal" co-simulation. In effect, any and every inference about the co-simulation system must be made by using only the sub-system inputs and outputs. Hence for the part of this text, surrogate models and other such approaches will be briefly touched in the future work section, but not covered in detail.

A note must be made regarding the use of the word "optimal" in the context of co-simulation time stepping. "Optimal" is not an applicable principle in this case, because optimality implies that there is a certain point at which a certain metric is at its minimum, maximum, or desired value. Co-simulation has no such metric that can be "optimized", and is not an optimization problem. Rather, it is a trade-off between the amount of real world time the simulation requires to execute, and the accuracy of the results. It is clear

that more accuracy requires more information about the system, which itself will only be available if more simulation steps are executed for the same amount of simulation time covered. This in turn inherently will negatively affect the amount of real world time needed to execute the simulation. “Optimal” in the sense of this text must be understood as the best trade-off, or the attempt to achieve a given or desired accuracy with the minimum amount of simulation calls possible. This definition itself already defies the “optimality” assumption, because the given or desired error is an arbitrary input to the system by the user, and hence can not be optimized by the system.

In addition to the above mentioned special issues, the general problem with linking dynamic sub-models together to from an overall integrated model for simulation is that all the issues that monolithic dynamic models face will also occur in an integrated model. However, due to the interactions of such sub-models, additional issues will arise that must be taken into account for proper co-simulation execution. Stability of the overall simulation must be considered the same way it needs to be considered in monolithic models. In addition to internal sub-model constraints, there may now exist additional inter-model constraints that must be met. Convergence between sub-models must be ensured. The principle of stiff systems becomes an issue as well, since the sub-models may exhibit vastly different dynamic behaviors that must be taken into account with respect to the impact onto the other participating sub-models. Singularities now may not only occur within states of the sub-models, but within the now integrated overall model. These issues may occur in a co-simulation even if the monolithic sub-models themselves behave perfectly well. Hence, the behavior of the integrated simulation can not be inferred or even predicted by simply looking at the sub-models only. As with monolithic models, the time stepping of the integrated co-simulation is of crucial importance for the stability and accuracy of the results. However, this must now take into account the different sub-models, their respective dynamic behaviors and the discretion over the variables that the integration engineer has available from the sub-models. This

opens up a new line of questions, namely, how to set the time step for the integrated model in such a way that the issues mentioned before are taken care of satisfactorily. While this subject has been discussed in the literature, the problem becomes more complicated when the equations of the sub-models are unknown and commonly applicable mathematical algorithms can not be applied any more in the straightforward manner they were initially designed for. The problem then becomes one of dealing with dynamic models whose "internals" are unknown or only partially known. The integration of these "Black Boxes" represents a new line of problems when considering adaptive time step setting.

4.1 Time Stepping In Co-Simulation

Time stepping within an integrated co-simulation environment inherently poses several issues that are unique to the co-simulation concept, and not found in monolithic models discussed above.

Since the sub-systems represent time-dependent systems, and since digital computers can not in a strict sense execute real continuous time problems, the system equations will need to be modeled and simulated in a time-discrete matter. It is true for all dynamic models (monolithic as well as coupled) that digital computers can not handle continuous time, and therefore need to calculate system states at certain discrete time steps. In essence, there are three different time scales to be considered: The real world "wall clock" time, which is the time that, simply spoken, the user has to wait until the simulation or time step is finished. It is one of the goals to reduce this time, as it has direct impacts onto the design phase duration of a system that relies or is based on computer simulation results. As discussed above, the time saving component is one of the major reasons for computer simulation, and the design engineer wants to take best advantage of this technology. Increasingly long simulation times will be contrary to the

intended goal. The second time is the “global time step”, commonly denoted as ΔT. This is the time step that covers the simulation time for the point where the sub-model(s) have received all input data and have been started, to the simulation time point where all model(s) have finished their calculations and have their results ready. After such sub-system calculations have been made, the co-simulation environment stops and exchanges the new data and system states between the sub-models. Then, depending on the intentions of the engineer, further simulation runs are performed. However, a given global simulation time step ΔT may be too large for the internal solver to provide useful and accurate results. Hence, the solver may choose to split the global time step ΔT into smaller “sub-time steps”. This can be done automatically or by user input. This smaller, internal solver time step is commonly referred to as “local time step”, denoted by Δt. It is clear, that Δt must be smaller or equal to ΔT, and must be an integer fraction of ΔT. Figure 32 depicts the general description of real world time, and simulation time steps ΔT and Δt. Figure 33 shows the execution timing for a co-simulation, as described above. In the context of this text, and for the development of the methods and approaches described herein, it is assumed that the local time step is set to be the global time step, $\Delta t = \Delta T$, unless otherwise noted. However, since the final application will be for “Black Box” sub-models whose behavior and solver are assumed to be inaccessible, this need not be the case for real word applications. However, the main point is that the sub-model must be executed correctly from the current time step T to the next time step $T + \Delta T$, and the actual approach of the solver to this problem does not affect the solution or method described here.

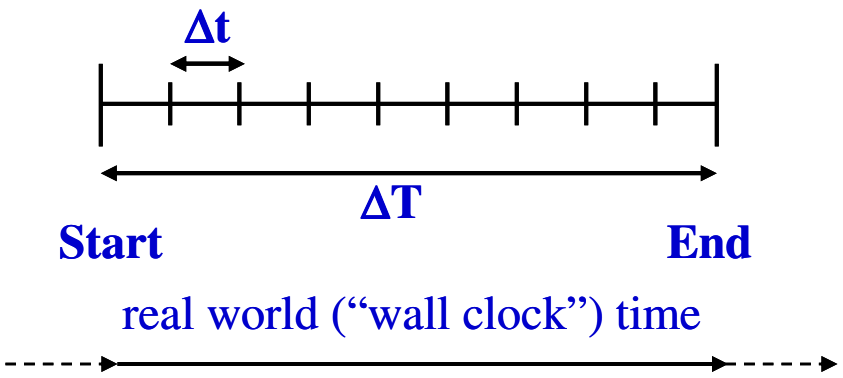

Figure 32. General description of real world time, global time step, and local time step

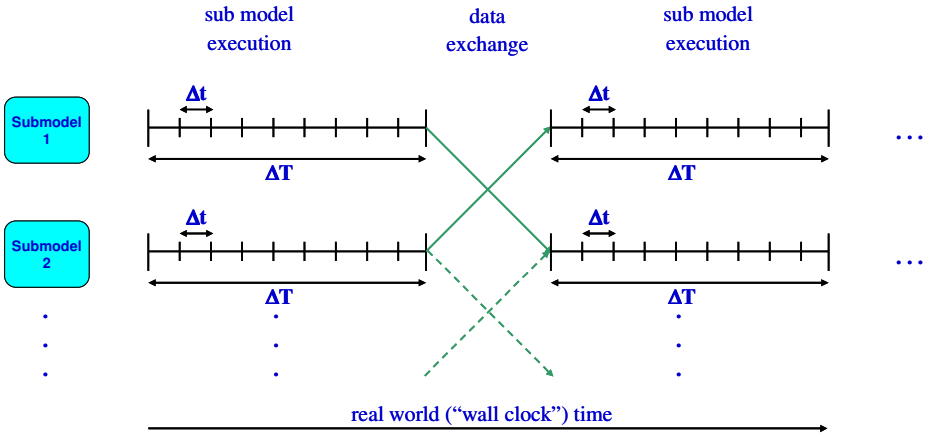

Figure 33. Timing for sub-model execution in a co-simulation

As can be seen, this time stepping schedule adds another “layer” of time steps to the basic case of numerical integration of differential equations. Basically, three different cases can be distinguished. The first case is the numerical solution of one single ordinary differential equation. This is achievable through the application of numerical integration algorithms, such as the RKF23 algorithm described above. Only one equation solver is necessary for this case. The second case is the solution of a set of ordinary differential equations. This means that several equations exist that need to be solved in parallel. Algorithms that extend on the simple algorithms, such as RFK23, have been developed and are able to solve such systems of differential equations. In this case, only one solver

is needed to solve the whole system. This setup would be the case for one single monolithic model that contains multiple differential equations within its model description. When executed, the single solver solves the set of ODEs in parallel, without necessary interactions from outside. The third case is when multiple single ODEs or sets of ODEs are distributed among different systems of ODEs. This is the case in co-simulation: The sub-systems represent single ODEs or sets of ODEs, which can be solved by one solver for each sub-model. However, the integrated overall system model then contains these single ODEs or sets of ODEs as sub-sets for the description of its own system. This implies that for each sub-system, there needs to be one solver, and for the whole co-simulation, the number of solvers is larger than one. This is the great difficulty when integrating dynamic systems into a co-simulation: The different equations can not be treated as one single set of equations that can be solved with only one single solver. Rather, the sub-systems must be solved with discrete solvers for each sub-system, and the overall model must then be integrated using the results from these discrete solvers. In other words, the integrated environment must provide a "super solver" that will integrate the different sub-models with an adaptive time step scheme, just like a single solver would integrate a single ODE or a set of ODEs. Table 1 depicts the three cases and the necessary number of solvers for each case.

Case	#ODEs	#Systems of ODEs	#Solvers
1	1	1	1
2	m	1	1
3	m	n	$n \leq x \leq (n*m)$

Table 1. Three cases of simulation

The resolution of the time stepping problem is the main part and contribution of this thesis. Specifically, the task at hand that this work is looking to propose a solution for is to determine the value of the global integrated simulation time step ΔT, in this case by using a proven algorithm for time stepping from the field of numerical integration. Figure

34 depicts the time step that is to be determined during the execution of an integrated co-simulated model.

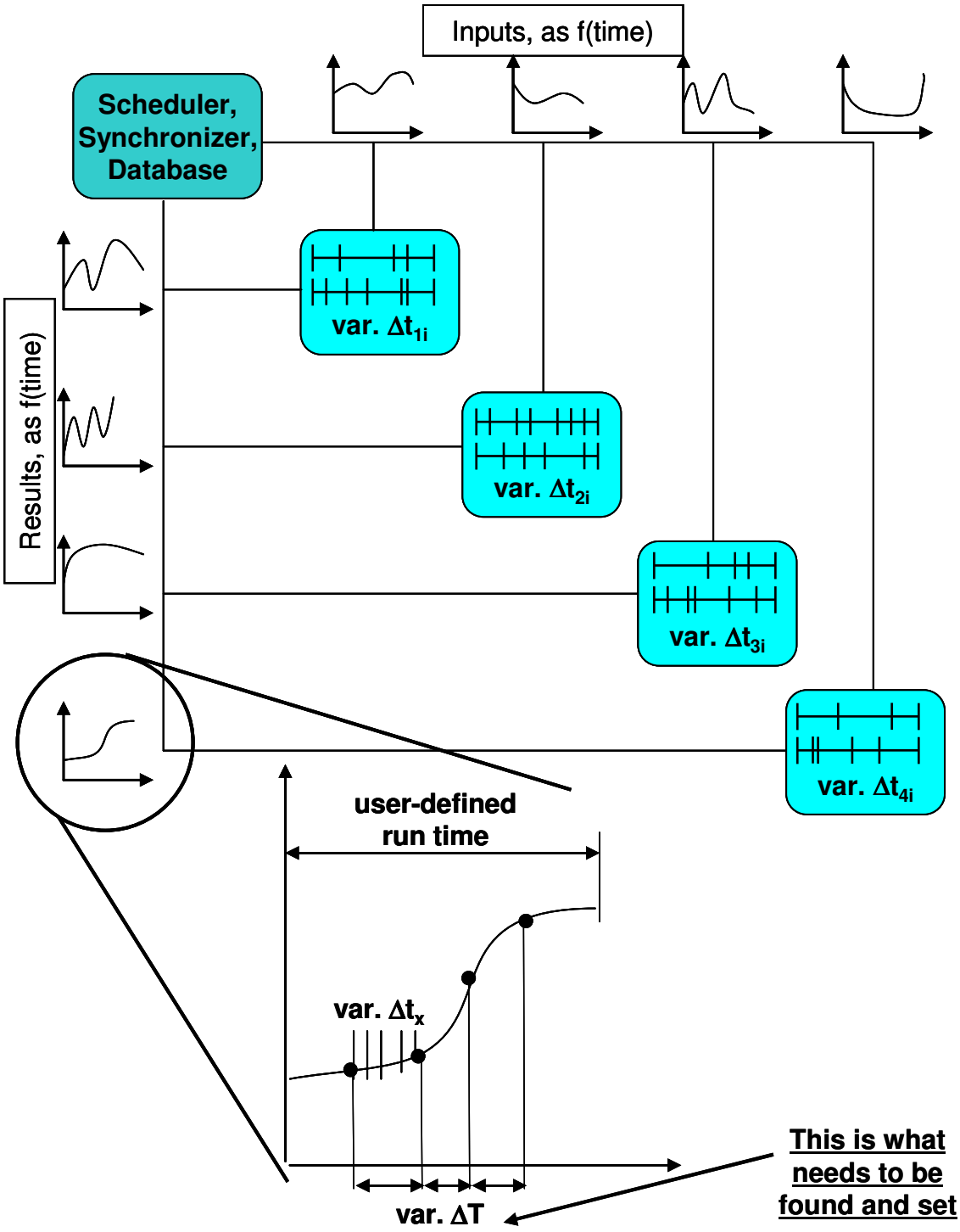

Figure 34. Time step to be determined in co-simulation setup

In the above figure, the time steps within the sub-models are denoted with Δt_{mn}, where Δt_m denotes the local time step within sub-system m, and n denotes the equation number within the sub-system, which must always be equal to or greater than one (there needs to be at least one equation to be solved within each sub-model). All the Δt_{mn} are solved by the internal solvers of each sub-model. Their internal time step is set by the solvers, not accessible or changeable from outside, and not the issue of this text.

ΔT is the time step at which all models stop their execution, and exchange their state variables between each other, as dictated by the scheduler. It is the time step of the "super solver" mentioned above. It is clear that the longer the models run between each data exchange (i.e., the higher ΔT is), the more the states will start to deviate. This in turn will lead to large changes in state variables during the exchange phases. Such changes can become so large that sub-models might not be able to cope with them properly any more. This will lead to malfunction of the integrated model. But even if this is not the case, as ΔT increases so will the deviations from the "real" state vector path(s). It is this problem that the applied algorithm for ΔT time stepping proposes to resolve. The new time stepping scheme from Figure 32 can then be extended and generalized as follows, see Figure 35:

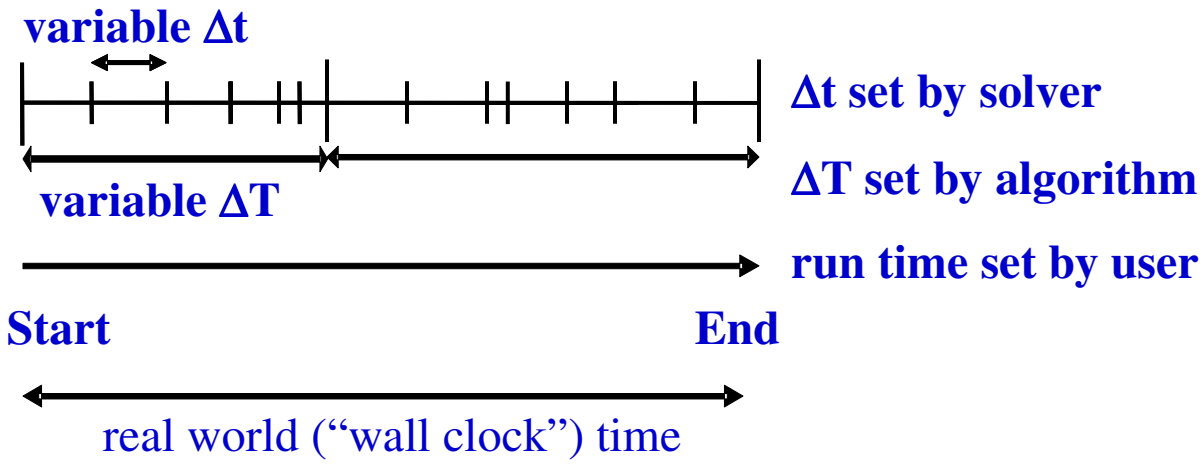

Figure 35. Time stepping scheme for integrated model execution

The run time is the user defined time for which the integrated model is to be run. It is essentially the observation time interval for which the user needs to have the model run to be able to examine the desired outcomes. ΔT is now the time step when the integrated model stops its execution and exchanges its data between the different sub-models. This is the time step that is being determined by the modified RKF algorithm proposed in this text. During each time step ΔT, the solvers for the sub-models will internally determine an "optimal" time step, using algorithms similar to RKF23, RKF45, etc. This time step Δt is not externally available, observable, or even changeable. This is not necessary, however, since the solution to the sub-models is best left to the respective solvers themselves.

As explained before, the proposed algorithm attempts to use methods from numerical integration of differential equations and apply them onto the problem of finding a global time step for a co-simulated dynamic system model. The rationale for this approach was explained, and is repeated here for clarity: In numerical integration of system ODEs, the path of the state(s) of a system must be approximated to lie within certain error bounds. The path itself is unknown, and must be deduced from information about current, and possibly previous states, and the system behavior. The algorithms exist and have been refined over many decades. The problem to be solved for the global time step in co-simulation is similar: The state(s) must remain within a certain error bound, and the way to do this is to vary the time step accordingly. The algorithms for numerical integration give a very good first idea of how to approach this problem for "Black Box" co-simulation, because in such a co-simulation case, no inferences can be made from system equations because those equations are unknown. Yet, the problem of error control must be solved, and thus numerical integration algorithms are used.

In order to investigate time stepping algorithms for dynamic system co-simulation, a toy problem was designed and set up, with which the application and

evaluation of different time stepping algorithms, was enabled. The toy problem has the advantage of being simple to set up and quick to run, enabling multiple test runs in a comparatively short time. Lastly and most importantly, it enables the user to compare the "real world" output to the output given by the time stepping algorithm. Using this toy problem, the proposed algorithm was implemented and tested, and further improvements realized.

4.2 Co-Simulation Setup and Method Application

Co-simulation in the widest sense describes the linking of several dynamic models into one overall integrated model. The actual underlying monolithic dynamic models (as described in Chapter 2) then become the sub-models of the integrated model. The advantages of co-simulation over monolithic models were discussed in Chapter 3.

The basic principles of co-simulation have been described previously. As concluded in the previous chapter, the method will now need to be put into a proof of concept, to check its general feasibility and shortcomings. To do this, a simple model must be chosen that can easily be set up in a modeling and simulation environment. The model's behavior must be known, as must be its underlying equations in order to model it. Ideally, the model would be set up as a monolithic model, thus enabling the verification of its behavior.

4.2.1 The double pendulum

One such model is the double pendulum. It possesses several properties that make it a very good test model. It is a dynamic model that is simple and easily understood. The underlying equations are readily available. It has been used in many applications, and results are easily being obtained from other literature and simulations. It is also non-linear, thus already providing for a test criterion for the method to be applied. The double

pendulum is comprised of an upper bar of length l_1 whose upper end fixed to a reference system via a rotational bearing. The lower end is equipped with a mass m_1. Attached to the lower end of the upper bar is the upper end of the lower bar, also via a rotational bearing. The lower bar has length l_2 and a mass m_2 at its lower end. Figure 36 shows a notional graph for a double pendulum.

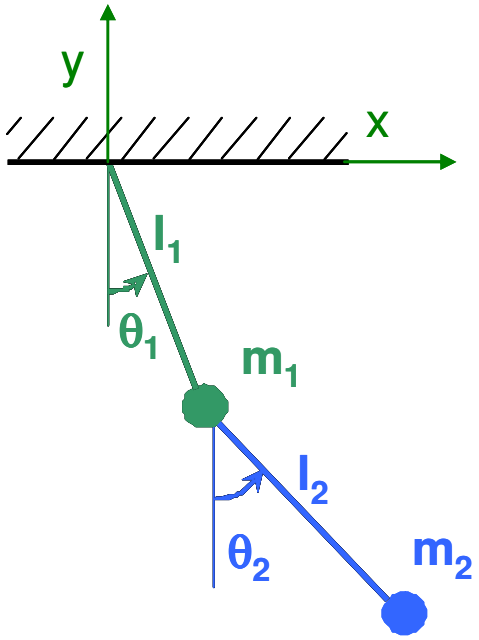

Figure 36. Double pendulum principal components

The pendulum model is a simplification of a real double pendulum, because it assumes the following:

- The two rotational bearings are frictionless
- There is no air resistance due to movement
- The two bars are have no mass
- The masses at the end of the bars are point masses

The pendulum was then modeled in Simulink™ by ASDL lab colleague Bassem Nairouz. The equations used are shown in Eq. 21. A small angle assumption is made in order to be able to simplify the equations and to get a closed form solution. The observed

variables used for the application of the method are the angular excitations of the bars, Θ_1 and Θ_2. The final model in Simulink™ is shown in Figure 37.

$$\begin{aligned} \ddot{\theta}_1 &= \frac{\omega^2 l(-\sin\theta_1 + M\cos\Delta\theta\sin\theta_2) - Ml(\dot{\theta}_1^2\cos\Delta\theta + l\dot{\theta}_2^2)\sin\Delta\theta}{l - Ml\cos^2\Delta\theta}, \\ \ddot{\theta}_2 &= \frac{\omega^2\cos\Delta\theta\sin\theta_1 - \omega^2\sin\theta_2 + (\dot{\theta}_1^2 + Ml\dot{\theta}_2^2\cos\Delta\theta)\sin\Delta\theta}{l - Ml\cos^2\Delta\theta}. \end{aligned} \tag{22}$$

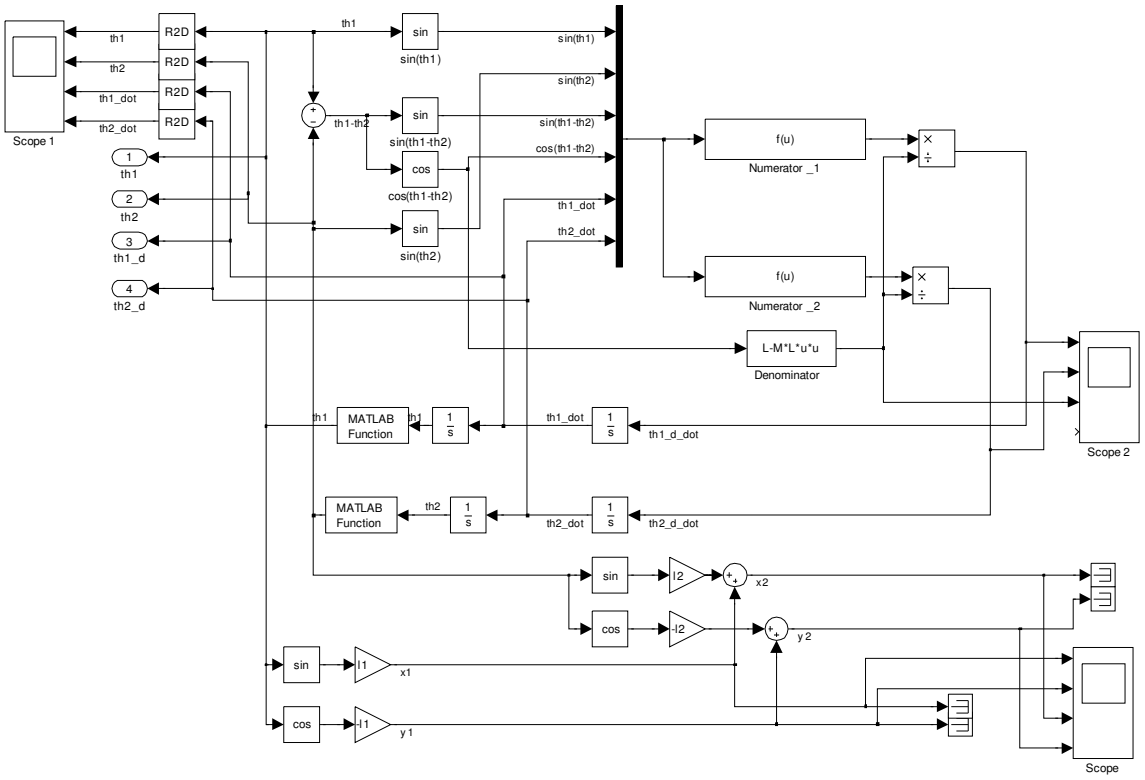

Figure 37. Double pendulum model in Simulink

The model was initially modeled as a monolithic model, meaning that it was one single model within the Simulink™ modeling environment. This enabled the testing of the model, and more importantly, it provided the basic model behavior of the model. This is due to the fact that all model equations are integrated within one single model, and only one solver is needed to integrate the equations during simulation execution. If the time step is chosen "sufficiently" small, then this model is the closest approximation to

the real world behavior a simulation can deliver. A "sufficiently" small time step was found by applying subsequently smaller time steps to the model until the deviation between subsequent time steps became so small that the time step was deemed "sufficiently" small with regards to deviation magnitude between the runs.

In order for this model to be implemented into a co-simulation environment, it had to be split into two sub-models, and recombined into a co-simulation. The natural way to do this is to make the upper bar and mass one such system, the lower bar and mass the other system. The split Simulink™ models are depicted in Figure 38 (upper pendulum bar) and Figure 39 (lower pendulum bar).

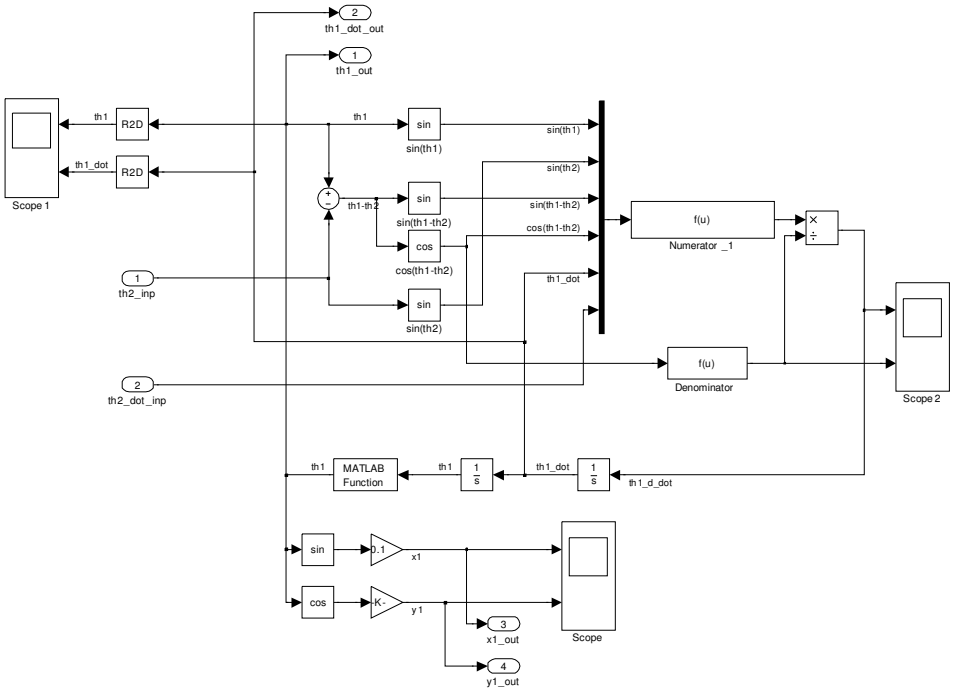

Figure 38. Double pendulum upper bar Simulink model

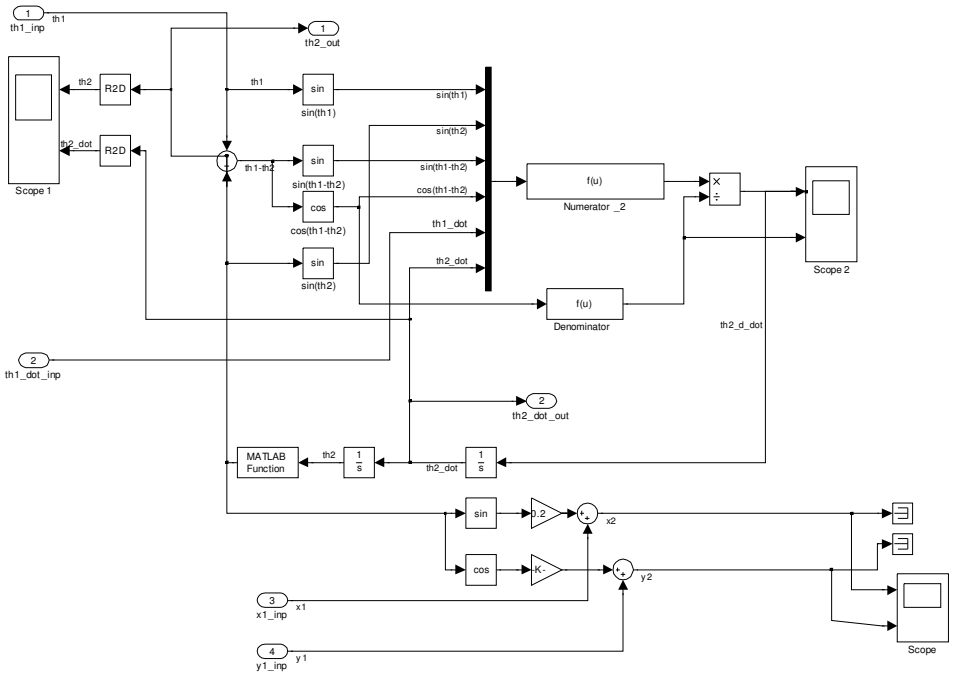

Figure 39. Double pendulum lower bar Simulink model

The integration for this simple model was not done in a commercial co-simulation environment, but rather in the Matlab™ programming environment. This made it very easy to access the Simulink™ sub-models, and program an effective scheduler. The notional setup is shown in Figure 40. The excitation angle Θ is used in later simulations as the state variable according to which the time step is set adaptively. This model was also used in the papers by Nairouz and Hoepfer (2008, 2009) described earlier.

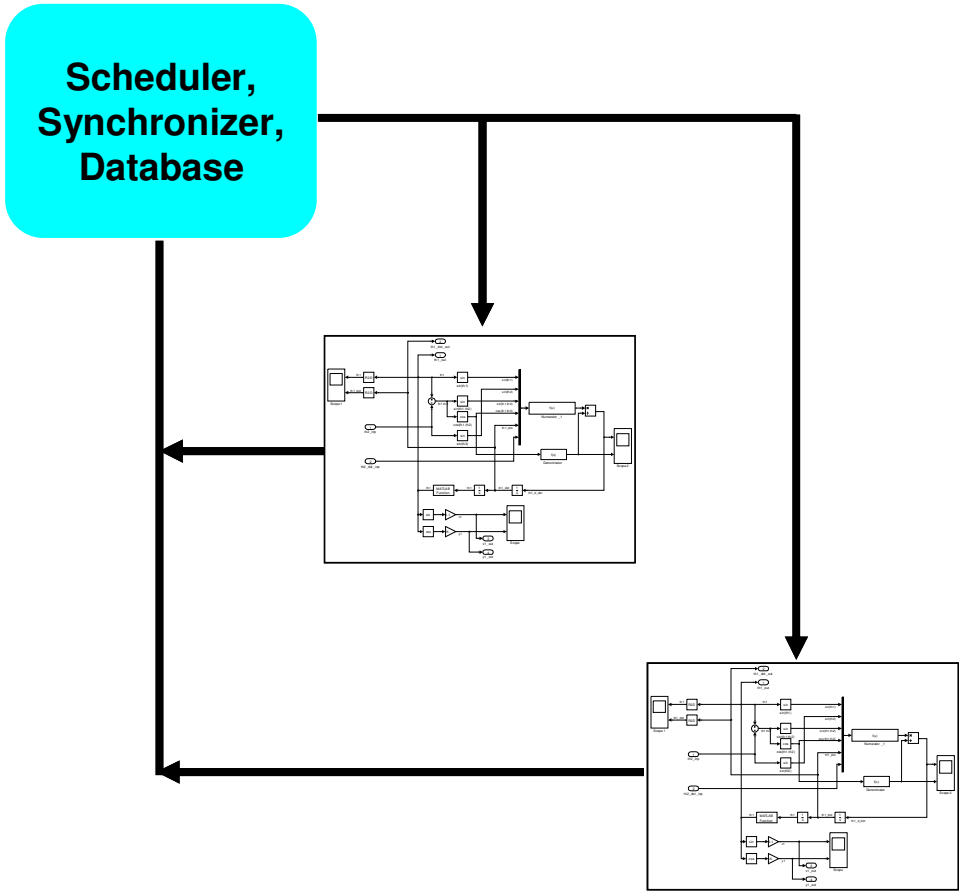

Figure 40. Double pendulum lower and upper bar integrated into co-simulation setup

4.2.2 Double pendulum co-simulation setup testing

4.2.2.1 Step 1: Compare monolithic and split model

Step 1 was then to compare the monolithic model and the split setup. As was mentioned, the monolithic model gets "as close as possible" to the real world model because it has all equations included and uses only one solver. This solver has all information available at all times. Thus, this setup gives the highest accuracy to the real world. Figure 41 depicts the comparison between the monolithic and the split model, both at small time steps. The graph shows the excitation angle in radians of the upper

pendulum. The excitation angles of the two pendulum bars are used as the exchange variables between the sub-models.

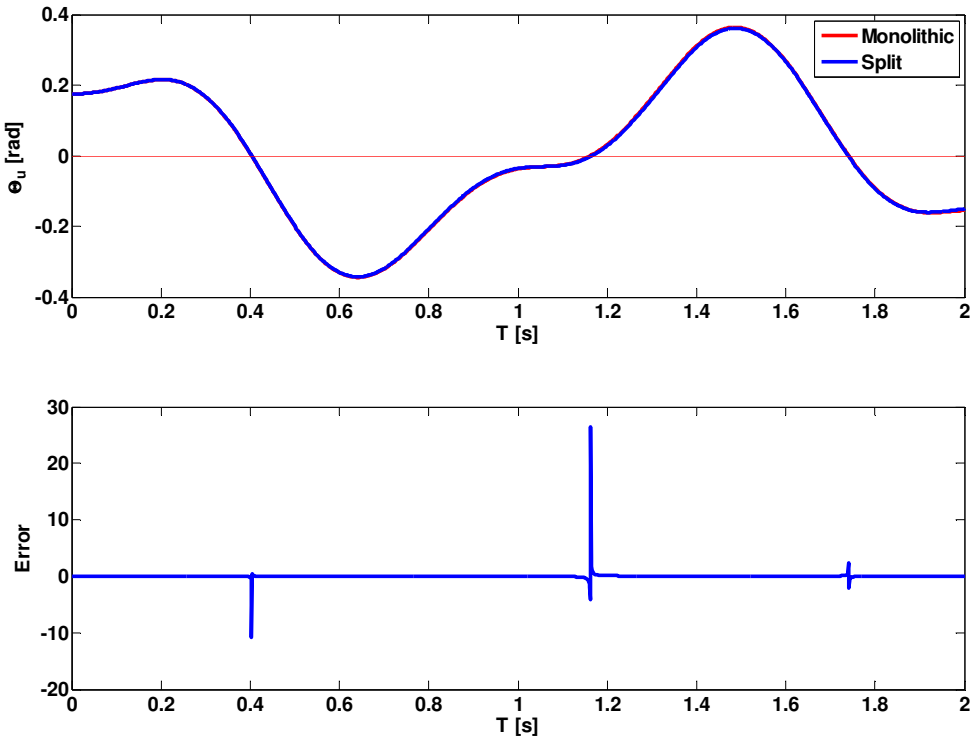

Figure 41. Comparison monolithic – split model, upper bar

The error, shown on the lower graph, has several spikes with a very large magnitude in it. This is due to the way the error is calculated. The deviation becomes very high when the values are very close to the zero line. Hence, this does not depict real error but rather is a result of the error calculation. Figure 42 shows the same comparison graph for the excitation angle of the lower pendulum bar.

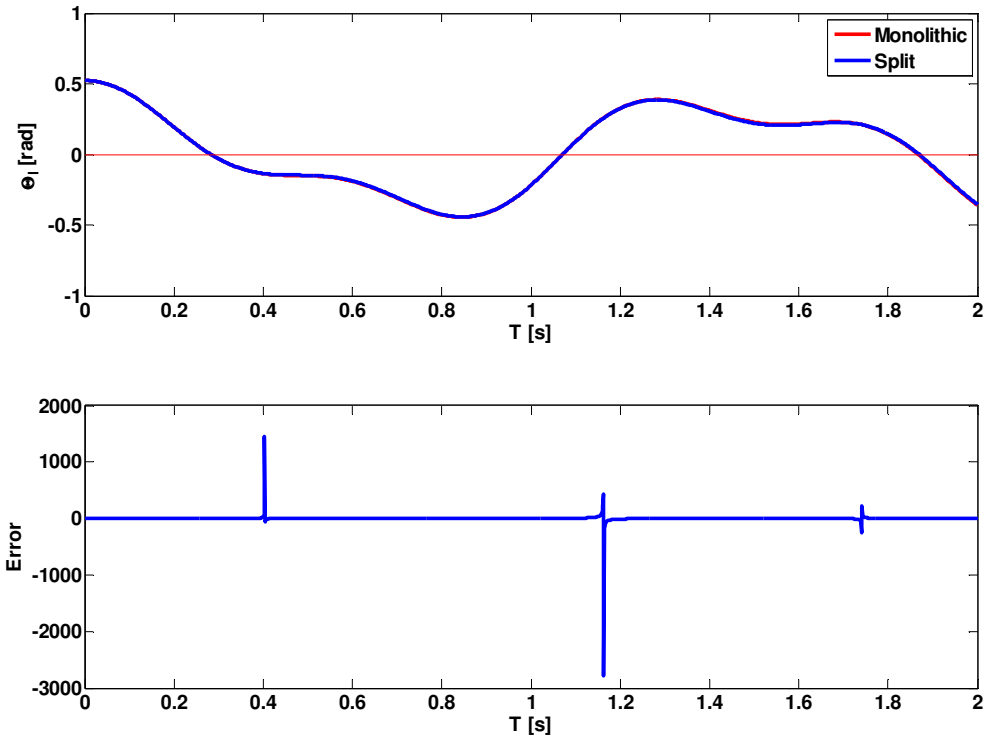

Figure 42. Comparison monolithic – split model, lower bar

The agreement between the monolithic and the split models is very good. Therefore, the split model is suitable for use for further investigations of the time stepping problem.

4.2.2.2 Step 2: Test for model time step stability

Step 2 was to test the split model for general stability against time stepping changes. Before applying a more sophisticated algorithm, it must be ensured that the simulation setup is inherently robust to time step changes. This will increase the confidence in any algorithm tested for time step setting. To do this, the simulation was run with a random time step that varied between 0.001 and 0.01. While the robustness of a co-simulation setup can not be inherently assumed for every situation, it is necessary

for this setup in order to be able to reliably test time stepping algorithms. Figure 43 depicts the curves for the upper pendulum bar for both runs, with a fixed small time step and a random time step. The second part of the graph shows the actual time step setting. It can be seen that the simulation is inherently robust against time step settings.

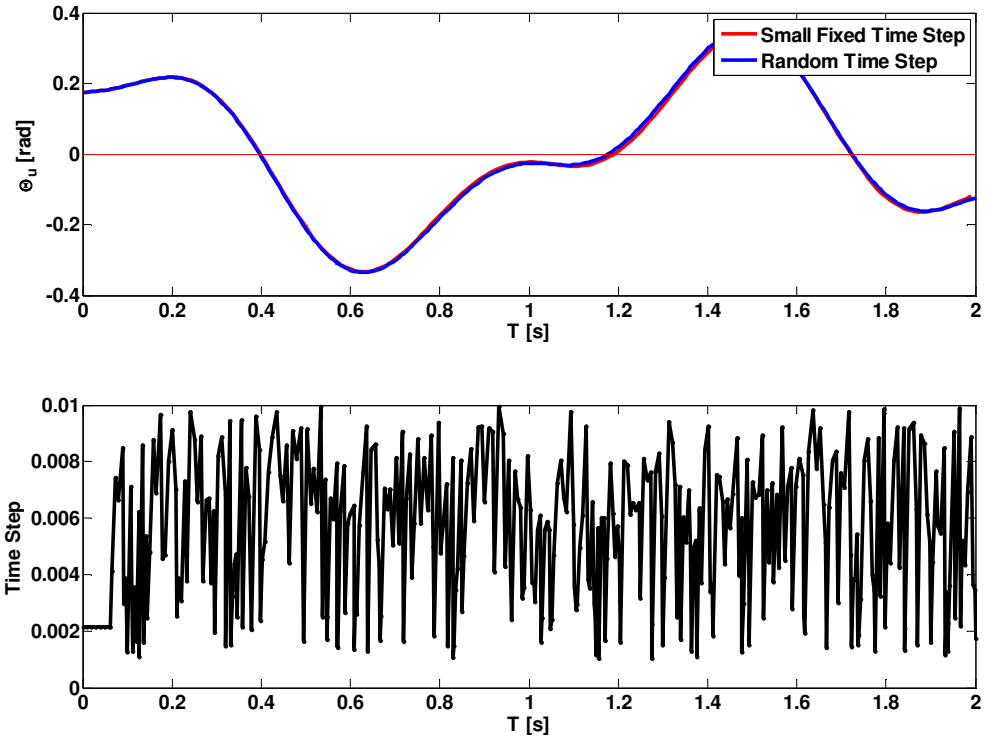

Figure 43. Test of integrated model for time stepping robustness

4.2.3 First attempt at time stepping with second derivative as parameter

During the derivation of the basic principles of numerical integration, the time step was always assumed to be constant, and set prior to execution of the simulation. No hint was given as to how the time step should be set, and what criteria might exist for finding feasible steps. As the discussion about the Euler method showed, a constant time step will result in increasing errors. If the aim is to reduce the error, or at least to keep it within a certain bound, then a constant time step will not be feasible. It is clear form the

Euler method discussion, that in order to control the error, the time step ***must*** be varied. This is because the change in error was proportional to the rate of change of the slope, as discussed in the Euler method section. Since the rate of change of the slope (the second derivative) is not constant, the additional error introduced at each step is also not constant. This can be changed when implementing a method that changes the time step according to some given error boundary. Some metric could be defined that determines the change of time step necessary for error control. From the earlier discussions, a self-evident metric would be the second derivative of the curve at the current time t. Intuitively this makes sense: If the curve changes quickly, in other words, if the rate of change of the slope is high, the curvature is high and in order to get accurate results, the time step should be small. On the other hand, if the curve changes only little or not at all, the time step can be large. Enright (1974) develops a class of second derivative formulas, investigates their stability for the solution of stiff ODEs, and implements them successfully in a variable order, variable time step method. In this case, the underlying equations are known. As described, the idea of local curvature by itself is insufficient, as it gives no hint as to how the second derivative translates into an effective time step. Figure 44 shows a notional curve and the application of a varied time step with second derivative as the input variable. Without loss of accuracy, the time step could be increased at section with constant slope (little curvature). The time step needed to be reduced at the sections with high curvature (rapid change of local slope).

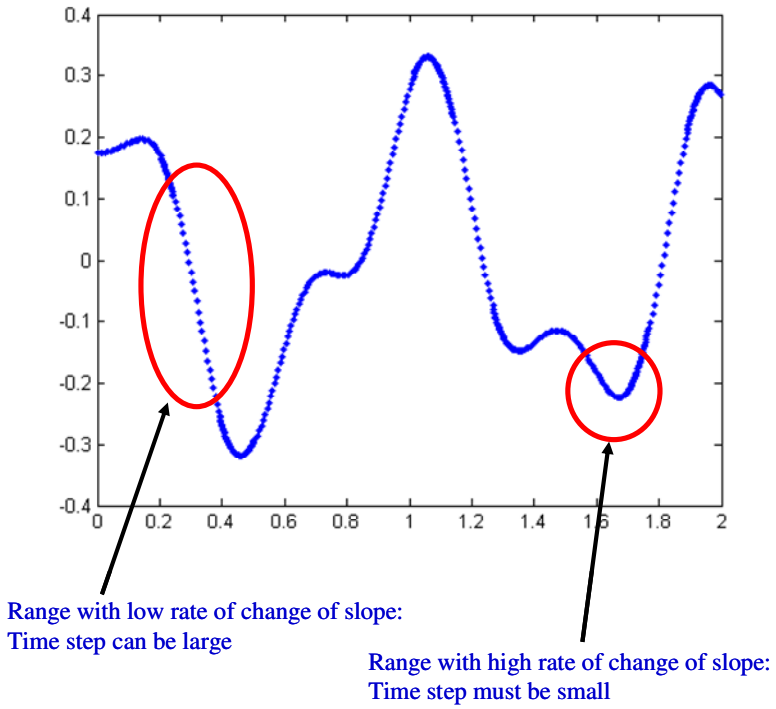

Figure 44. Notional explanation of impact of second derivative on time step

Nairouz and Hoepfer (2009) presented a notional solution to this approach. Using the rate of change of the local slope (second derivative) as a "trigger" criterion, the time step was adapted to the curvature of the underlying function. Thresholds for time step changes were arbitrary values of the second derivative. The upper and lower time step limits were also chosen arbitrarily. Hence, there was no error prediction or binding. Also, the chosen trigger thresholds of the second derivative, and the time step limits were chosen arbitrarily, for the model used in the paper, just to show how this method could be employed. For any other model, these parameters would have to be adapted accordingly. This means that the method presented in the paper was just a proof of concept, no scientific approach with solid foundation. It also does not help to make any time step selections for arbitrary models. Hence, there is room for improvements. Figure 45 shows a notional result for a certain combination of second derivative thresholds and time step limits.

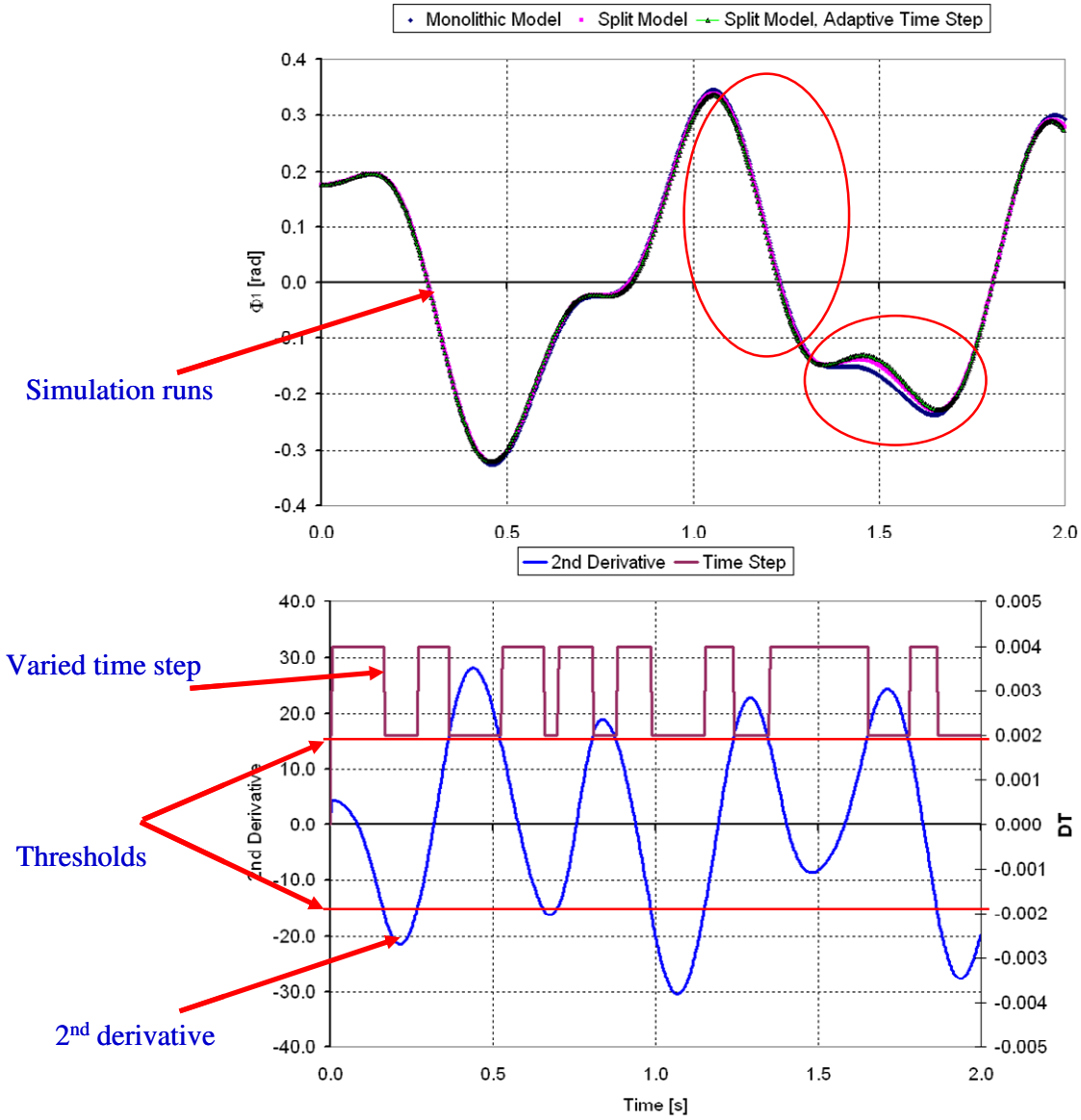

Figure 45. Applied time step variation with second derivative method

The described method needed the user to set the parameters and limits according to her understanding about the systems, or some sort of "gut feeling". But to select a proper time step for any given or underlying model, a more solid method must be employed. There are a variety of such methods, all developed for the solution of numerical integration of DEs. These methods have been widely employed, and enable to get rid of parameter arbitrariness and allow for error binding within a certain given

tolerance. While there are many different such methods to choose from, for current first approach development of a methodology, the focus will be on the Runge-Kutta-Fehlberg method, commonly abbreviated as RKF23. Other methods worth mentioning are Adams-Bashforth and Adams-Moulton. A specific extension of the Runge-Kutta family of algorithms was developed by Cash (1976). It presents an A-stable, semi-implicit Runge-Kutta procedure requiring at most one Jacobian evaluation per time step for the approximate numerical integration of stiff systems of ordinary differential equations. It also includes a simple procedure for estimating the local truncation error and derives efficient integration procedures using the error estimate. Such enhanced algorithms can serve as the basis for the development of own methods for the specific application to co-simulation as defined within the context of this thesis.

A short note should be made here regarding the use of a derivative for time step settings. The use of a derivative would indicate that the magnitude of the curve itself does not matter. The initial methods to solve ODEs and set a time step adaptively, were based on (ordinary) differential equations of the form Eq. 1. This equation indicates that the slope at the current point is a function of the current time, system state, and possible external inputs to the system. But this is only due to the fact that the underlying algorithms are based on such ODE forms. The actual co-simulation model however, will have a slope that does not depend on its current magnitude (we leave external inputs out of inspection for now, since they will have completely different impacts on the model, and hence would need to get “special treatment” by any algorithm). Instead, the current slope and its rate of change will be calculated based on previous, current, and possible future system states. These states are taken into account only by their differences when calculating the slope and curvature. Similarly, the time steps for these points are taken into account only by their differences, not their absolute values. This leaves only the current x-coordinate, i.e. time, as the independent variable for the evaluation of the current slope and curvature. If such an approach turns out to be true, any method that sets

the time step solely on current derivatives would then be “universal”, i.e. it would not matter what current time step or current magnitude the state variable has. This would greatly extend the usability of such a time stepping algorithm. This text will investigate further whether derivative based algorithms can be a feasible and viable time step setting metrics.

4.2.4 Application of RKF23 as time stepping algorithm in “Black Box” simulation

4.2.4.1 Step 3: Programming and testing the RKF23 algorithm

Step 3 was to program and test the underlying RKF23 algorithm with defined mathematical functions. This was described in Chapter 2, and depicted in Figure 20, Figure 21 and Figure 22.

4.2.4.2 Step 4: Apply RKF23 algorithm to split model, single bar as reference

In Step 4, the double pendulum model was then run using the RKF23 adaptive time step method introduced in the previous chapters. The model is run from user-determined initial conditions, which in this case translates to a certain initial excitation angle and no initial angular velocity. For this simulation, local time step and global time step were set equal, $\Delta T = \Delta t$. Therefore, no internal time stepping takes place within the models during an execution step. As a first attempt, the time step was set according the angular excitation of one bar only. This means that only one variable was monitored and used to map its second derivative to an appropriate time step for the overall simulation. Figure 46 shows the result for the case were the upper bar excitation was used for the time step mapping, Figure 47 shows the same result for the lower bar.

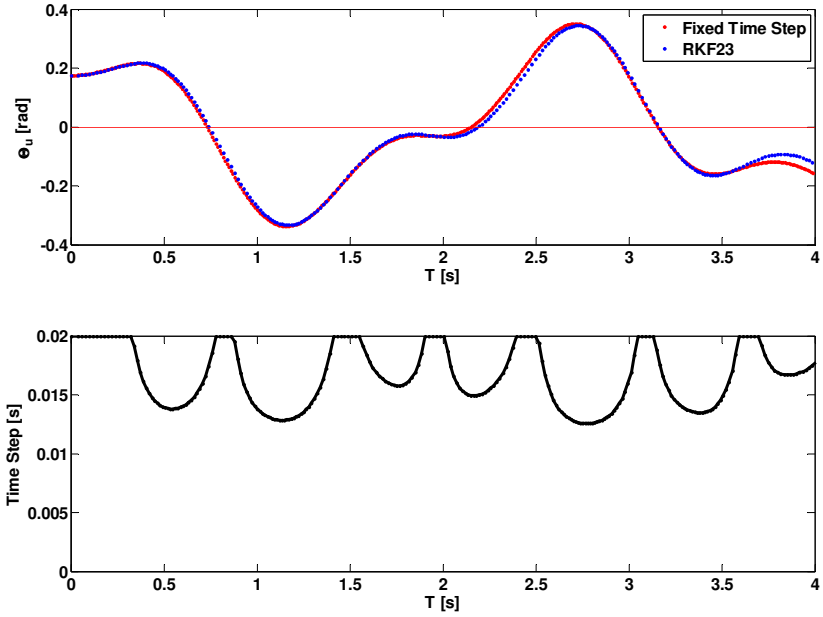

Figure 46. Applied RKF23 algorithm to double pendulum model, upper bar as reference

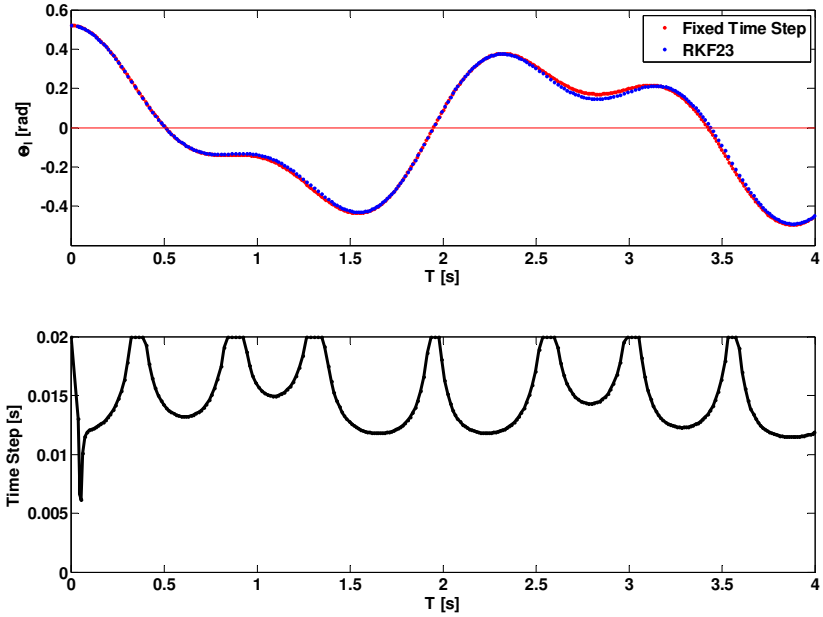

Figure 47. Applied RKF23 algorithm to double pendulum model, lower bar as reference

This also serves as the verification that the split model with time stepping runs accurately compared to the split model when run with very low time step as a comparison basis. In both graphs, the green constant line is the comparison curve for the known system behavior (the split model was run with very low time step and hence is accurate). The dotted blue line is the result from applying the time stepping algorithm to the simulation. The results can be seen to be very accurate, and following the underlying accurate base model very nicely. The black line shows the time step resulting from the mapping of the second derivative. (The sharp drop in time step in Figure 47 [lower bar] is due to the fact that the simulation starts off with the highest allowed time step. This occurs at a region where the time step should be low [the change in slope is high]. The algorithm immediately accounts for this and tries to adapt the time step accordingly; this is nice evidence of the robustness of the algorithm).

A note must be made regarding the calculation of the slopes, which is necessary for algorithms such as the RKF23, as has been described in Chapter 2. Since it is assumed that the underlying sub-models are "Black Boxes" which only have state variables as outputs, the slope is not readily available as it would be when solving an ODE. Hence, it must be derived from the state variables themselves. In the current application, this was done using a four-point polynomial approximation. The points used are the previous point in the time history of the state variable, the current point, and the two future steps (half-step and full-step point) that are calculated anyway when executing the RKF23 algorithm (see Figure 48 for a notional depiction of a state trajectory curve and a fitted third-order polynomial fitting line).

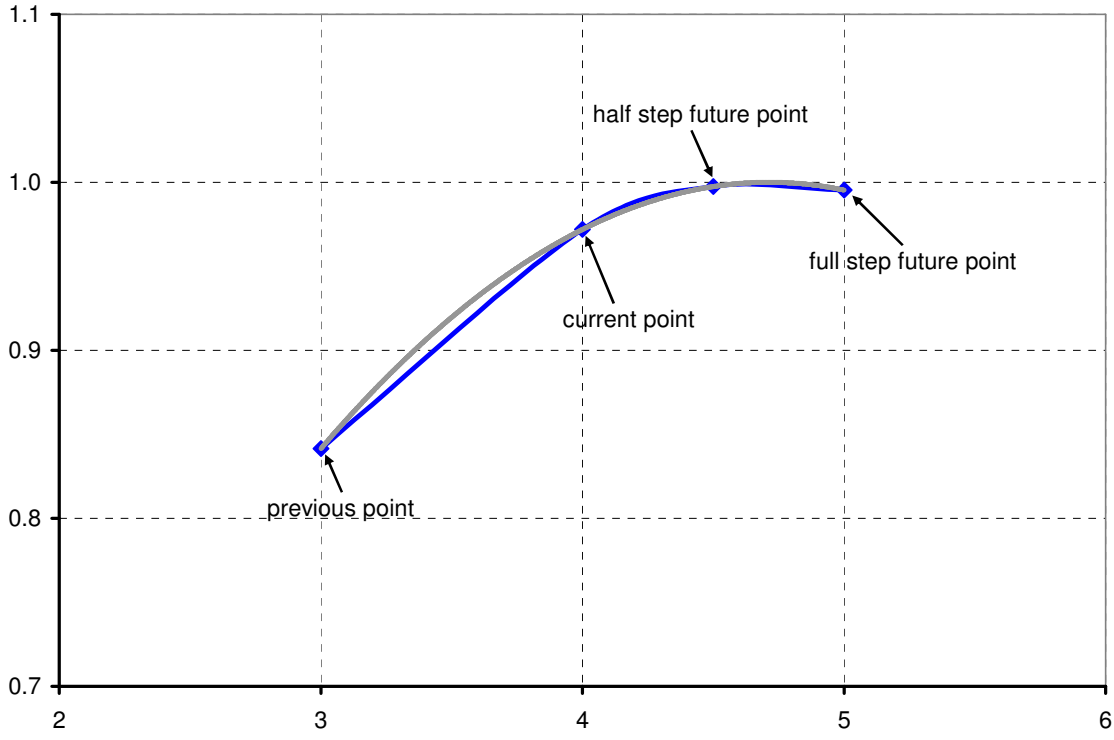

Figure 48. Fitted third order polynomial for slope determination

Since the simulation can not calculate points themselves, like e.g. an ODE would, the simulation must be run from the current point to the half-step point once, and once more from the current point to the full-step point, both with equal initial conditions. From the four points obtained through this approach, a polynomial with degree 3 could be constructed using the Matlab command "POLYFIT". From this, the derivative can be readily taken at the three points needed. The reason why a polynomial of degree 3 was chosen is the following. The RKF23 algorithm calculates the difference between the new points calculated using the midpoint slope and the new point calculated using the slopes from the current point, the mid-point, and the full-step point. If a polynomial of degree 2 is used, then the two new points will be identical, because the distances between the future points is equal and the slopes will not change. This means that the changes between subsequent points will be equal as well. Hence, the algorithm will not detect any error, and thus keep the time step at its highest possible value at all times. This is clearly

wrong. By using a polynomial of degree 3, the slopes will change between the points, and thus there will be a discrepancy between the calculated points. This error will correctly be used to determine the new time step. However, the pendulum model allows for the output of the slopes from the Simulink model itself. The slopes are calculated within the model, and were made as connections in the model such that they could be read out directly from the model at each time step. This enabled a direct control as to whether the calculated slopes are correct or not. Figure 49 shows the comparison between the calculated slopes at the three points, and the slope at the current point as per direct output from the model. The agreement between the calculations and the actual slope is very good. The lower graph shows the discrepancy between the current point slope calculated and from the model. The agreement is very good. As before, the error is high when the slopes are close to the zero line; this is due to the way the error is calculated, and does not mean that the discrepancies are higher there.

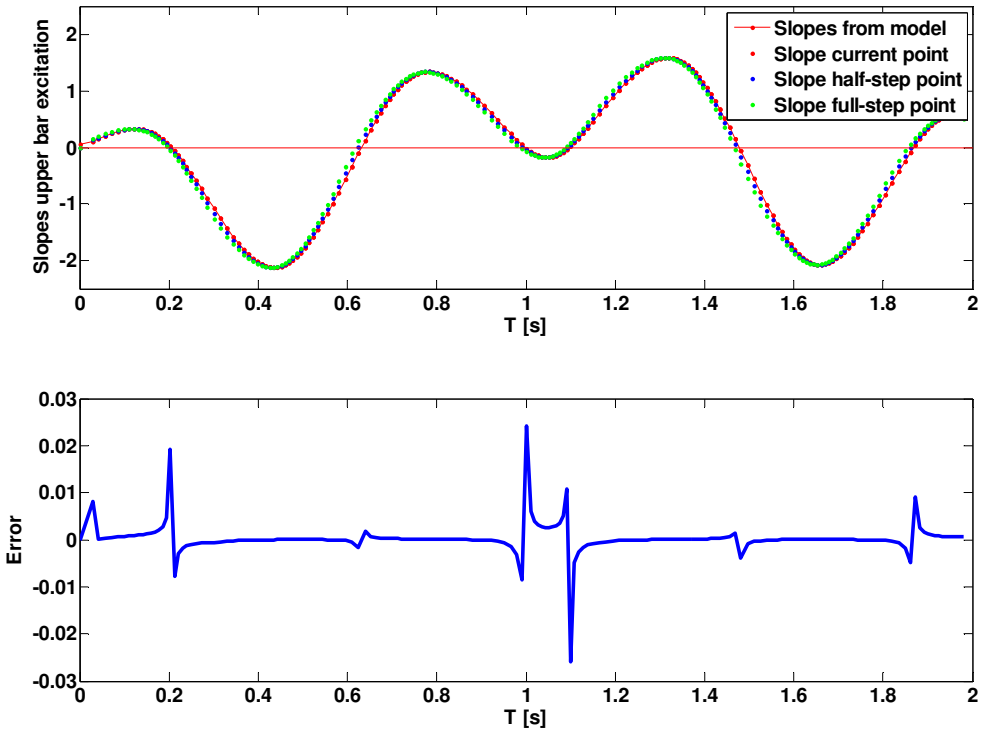

Figure 49. Comparison between calculated slope and slope acquired directly from model

When looking at the RKF algorithm as described in Appendix B, one can see that it is computationally quite expensive. For every point, the simulation must be executed at least three times (one time for the actual point, one time for the half-step point, and one time for the full-step point). Compared to a fixed step algorithm, this is justifiable only when the amount of time steps used by the RKF algorithm is lower than the amount of time steps for a fixed step algorithm. Therefore, the time stepping count was evaluated for both algorithms. Number of function calls as a metric for determining the efficiency of the algorithm is justified elsewhere in this text.

4.2.4.3 Step 5: Test RKF23 algorithm for different dynamics

In Step 5, the double pendulum model's advantage of allowing for easy change of model dynamics was used to test the model's behavior for this situation. In order to calculate the amount of time steps needed for different comparison cases to the fixed step setup, the system was run with a "slow" and a "fast" setup. The physics of the pendulum state that the frequency of the pendulum is a function of the length of the pendulum bar. Therefore, by changing the pendulum bar lengths in the model, a "fast" and a "slow" case could be run and the amount of time steps needed could be calculated. Figure 50 shows the graphs for a "slow" system (pendulum bar lengths = 1m), Figure 51 shows the same system with a "fast" setup (pendulum bar lengths = 0.5m). Both graphs show the upper pendulum bar graph only.

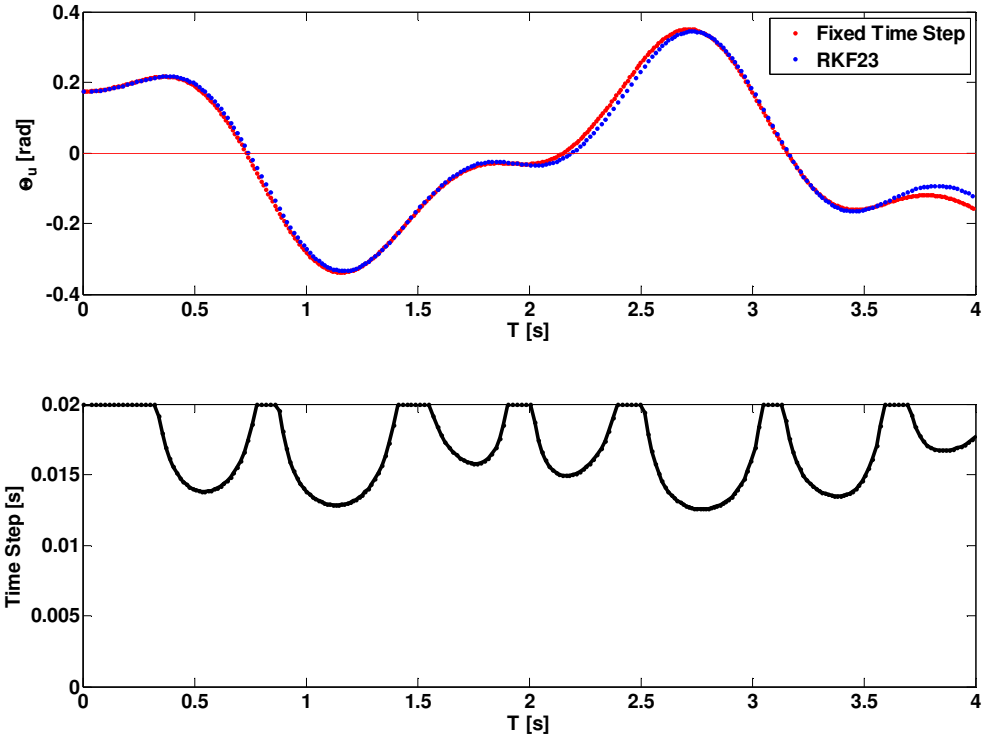

Figure 50. Double pendulum model "slow" configuration, upper bar

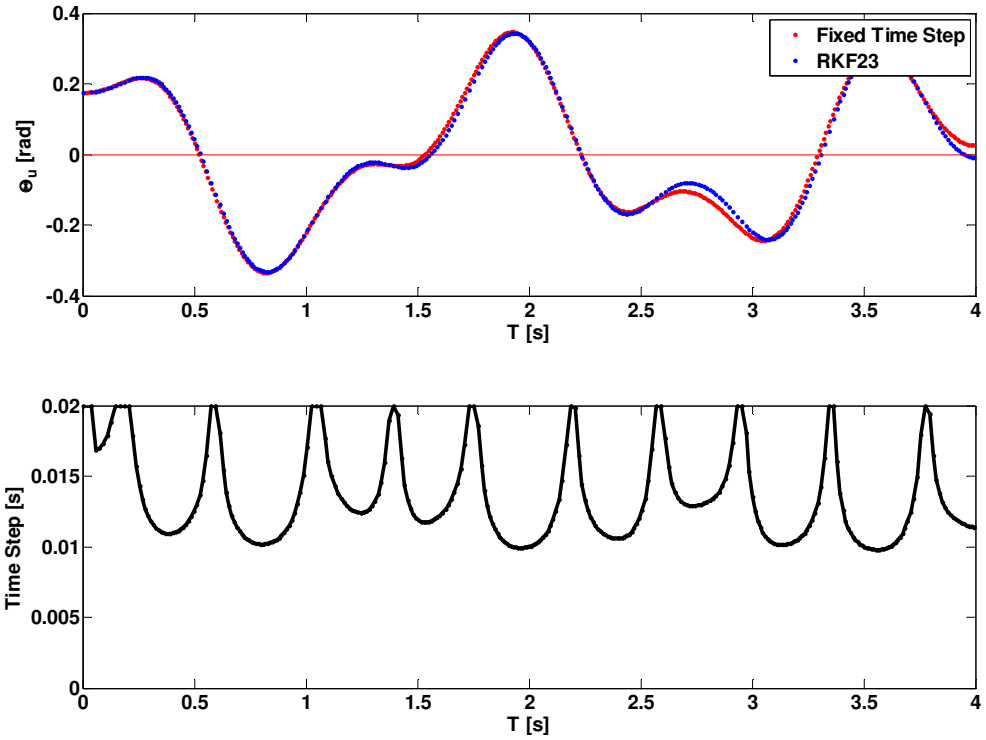

Figure 51. Double pendulum model "fast" configuration, upper bar

It can be readily observed that the "fast" system is a time compressed version of the "slow" system, with equal behavior but shorter wave lengths. It is also readily observable that the time steps are decreased more often in the "fast" case, and to lower levels than in the "slow" case. Both situations make intuitive sense, and serve as a confirmation of the validity of the model and the algorithm.

The general equations for the pendulum frequency show that

$$1/f \propto \sqrt{l} \tag{23}$$

with f as the pendulum frequency, and l as the pendulum bar length. This in turn means that for frequency f,

$$f \propto 1/\sqrt{l} \tag{24}$$

holds. The ratio of the frequencies between the "fast" and "slow" case can be verified in Figure 50 and Figure 51. For example, the first minimum in Figure 50 is at about 1.2 seconds, the first minimum in Figure 51 is at about 0.85 seconds. Since the pendulum bar lengths have been halved, the proportionality factor is SQRT(2) = 1.41, which is 1.2 seconds / 0.85 seconds, the ratio between the two frequencies. The number of function calls required for the two cases (using the upper pendulum excitation as the investigated state variable) can be seen in Table 2:

	Fixed Step	**RKF23**
"Slow" case	1600	1478
"Fast" case	2260	1856

Table 2. Number of function calls for "slow" and "fast" dynamics cases

Before continuing the discussion about these results, a short side note must be made with respect to choosing the number of function calls necessary as a metric to judge the efficiency of the algorithm. Each function call requires the initialization of the sub-model, its execution, trajectory interpolation (covered later) and slope approximation, and the exchange of the data. Of all these aspects, the actual execution of the sub-model is the computationally most expensive part. This computational expense also translates directly into real world time requirements and thus, cost. The other issues mentioned that need to be taken care of for each time step (the computational overhead) are much less demanding in terms of computational expense. Therefore, in order to evaluate the performance of the implemented time stepping algorithm, the number of function calls is the best metric to determine the efficiency of this implementation. This issue is further discussed under "Computational overhead" in Chapter 5.

The number of time steps in the fixed case was increased by the factor SQRT(2), as necessary and explained above. The application of the RKF23 algorithm shows that it requires less function calls despite the fact that for every point, the RKF23 algorithm

requires three function calls, compared to one function call for the fixed step algorithm. Therefore, it already represents an improvement over the fixed step algorithm.

However, the point could be made that with careful system observation, a fixed time step could be found that might outperform the RKF23 application case. This though is only possible if it is known in advance that the dynamics will not change significantly during the simulation execution run. If this is not the case, and the dynamics of the system can not be predicted in advance (as is the case most of the time), then the adaptive time step plays out its full potential, demonstrated as follows.

The double pendulum model allows for easy change of the dynamics, as mentioned before. This is also true when the simulation is already running. In other words, the model allows for changing dynamics during the simulation execution. This is demonstrated in the following case. The simulation was run in the "slow" setup for one second simulation time. Then, at simulation time T = 1s, the dynamics of the pendulum were toggled to "fast" for the duration of one second simulation time. After that, the dynamics were changed back to "slow". Figure 52 shows the graph for the upper pendulum excitation angle for the case of toggled dynamics.

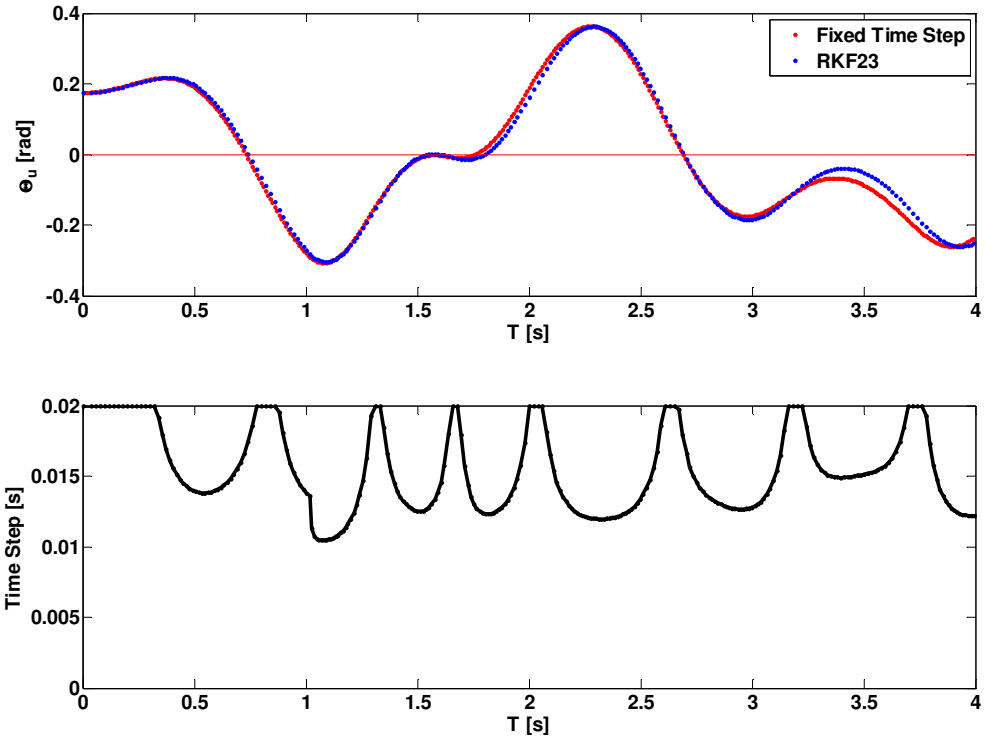

Figure 52. Double pendulum model "slow", toggled to "fast" at 1s < T < 2s, then "slow" again, upper bar

From the graph, it can be observed that the period time is changing during the period when the dynamics (= the pendulum bar lengths) are toggled. This becomes even more evident when looking at the time step graph. The adjustments become more frequent, showing good reaction of the algorithm to the system behavior.

In the case described above, if the system dynamics are known before the simulation execution, the application engineer would be forced to set the time step in such a manner that it safely covers the fastest dynamics occurring during the simulation. In the case presented here, the time step for the whole simulation time would have to be chosen for the "fast" case setup, even though the simulation clearly has extended time sections where the dynamics are in the "slow" regime. Hence, the use of a fixed time step in such a case creates a huge overhead of function executions that are unnecessary when

using an adaptive time stepping algorithm. Table 2 from above can now be extended by the case of toggled dynamics, see Table 3:

	Fixed Step	**RKF23**
"Slow" case	1600	1478
"Fast" case	2260	1856
Toggled case	2260	1604

Table 3. Number of function calls for "slow", "fast", and "toggled" dynamics cases

It can clearly be seen how the adaptive time step uses even less function calls now. This is of course due to the fact that in the areas with slow dynamics, the algorithm can increase the time step, thus reducing the necessary amount of function calls. Only when the dynamics become faster, shorter time step intervals require more function calls and time steps. Once the dynamics become slow again, the amount of time steps and function calls reduces again. The RKF algorithm thus not only helps to control the simulation error but also to set a time step according to the requirements of the simulation.

4.2.5 Extension of RKF23 application to co-simulation

4.2.5.1 Step 6: Apply algorithm to all state variables, using minimum of all time steps

It can be clearly observed from the time series plot of the state variables that the time step behaves as required: If the local curvature is high (the second derivative is high), the time step becomes low, and vice versa. This confirms the general idea of the applied algorithm. However, this type of simulation and application of the method is not fully implemented. It was applied as a first test to check whether the method works as figured. The reason why this is an incomplete application of the algorithm is that as stated above, only one state variable was used for the different runs as the metric to which the

time step was mapped. In essence, this approach was that of only *one* dynamics system with only *one* state variable under investigation. In a co-simulation however, *every* state variable that is exchanged between *all* the sub-models must be taken into account. This is easily understood if one takes the case of one state variable having slow local dynamics, the other one having fast local dynamics. The state variable with the slow local dynamics would allow for a large time step, while the state variable with the fast local dynamics would require a small time step to keep the accuracy high. Hence, in a co-simulation all the state variables must be considered by mapping the time step to their local curvatures and choose the smallest time step that results from this approach. Figure 53 shows the notional setup of a co-simulation, making clear that all the simulation states need to be considered. Figure 54 shows how the smallest time step is then chosen: From the current point in time (which is the same for every state variable), the next time step is determined by mapping it to the local curvature of all the state variables exchanged. The one time step that comes out to be the lowest will then be used for the overall simulation, and all the sub-models to be executed.

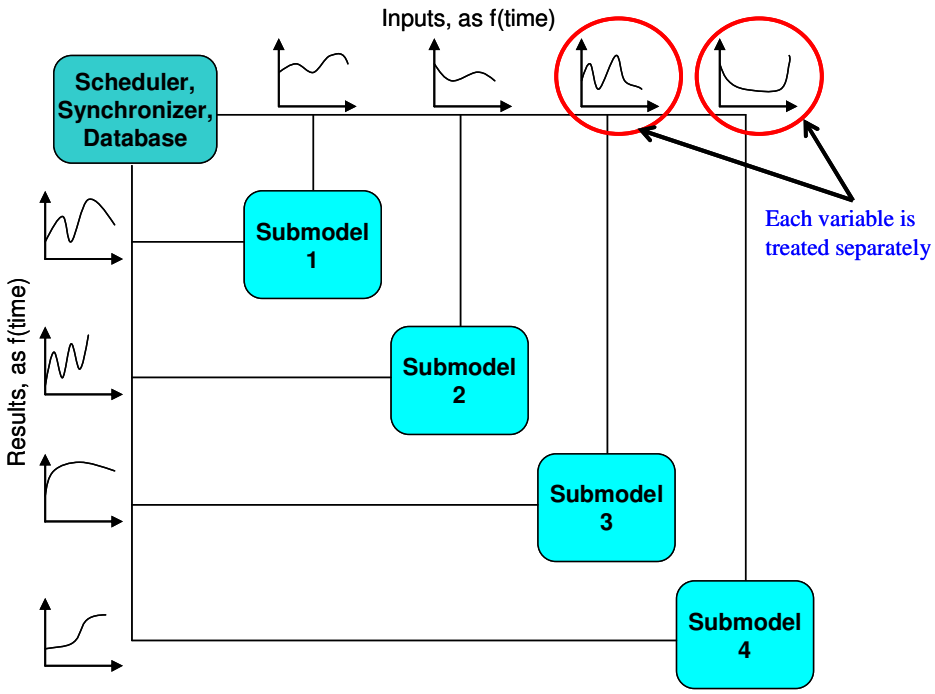

Figure 53. Notional depiction of the variables under consideration as time stepping metrics

As has been discussed before, each one of the state variables that are sub-model outputs and needs to be exchanged with other sub-models is an output from a dynamic model, and feeds into the overall integrated model, which itself is dynamic model. Hence, all the state variables exchanged must be taken into consideration when exchanging the data. All of these state variables must be examined with respect to the overall time step ΔT for the co-simulation. The time step will then be determined for all models, and with respect to the state variable that shows the fastest dynamics. Figure 54 depicts how this is done in a notional way.

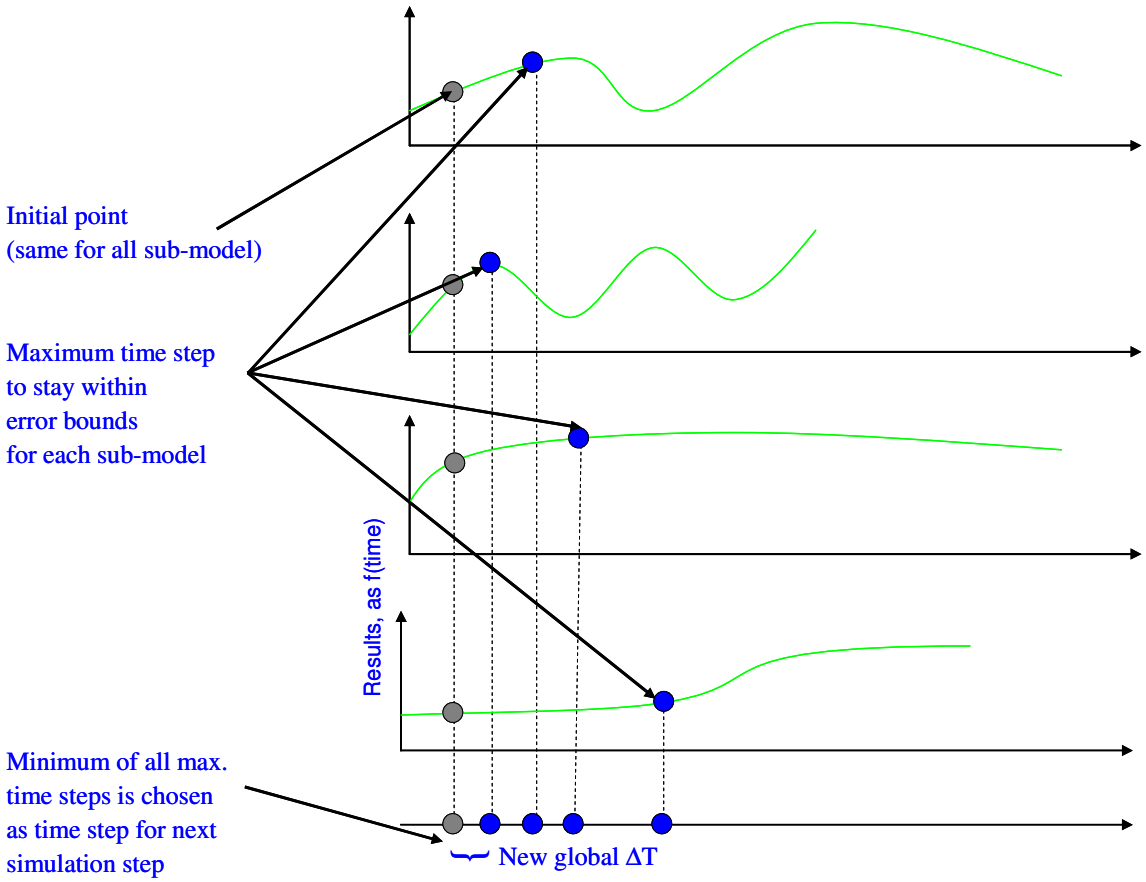

Figure 54. Notional explanation of new time step determination approach

This is now implemented into the Matlab algorithm. The simulation runs the code for both pendulums and determines which one of them requires the lower time step (has the faster dynamics). This time step is then adapted to the simulation as the overall simulation time step ΔT. Figure 55 shows the result for "slow" dynamics, Figure 56 the result for "fast" dynamics. Figure 57 shows the result for the case where the dynamics were switched from "slow" to "fast" and back, as a control case for the adaptability of the algorithm.

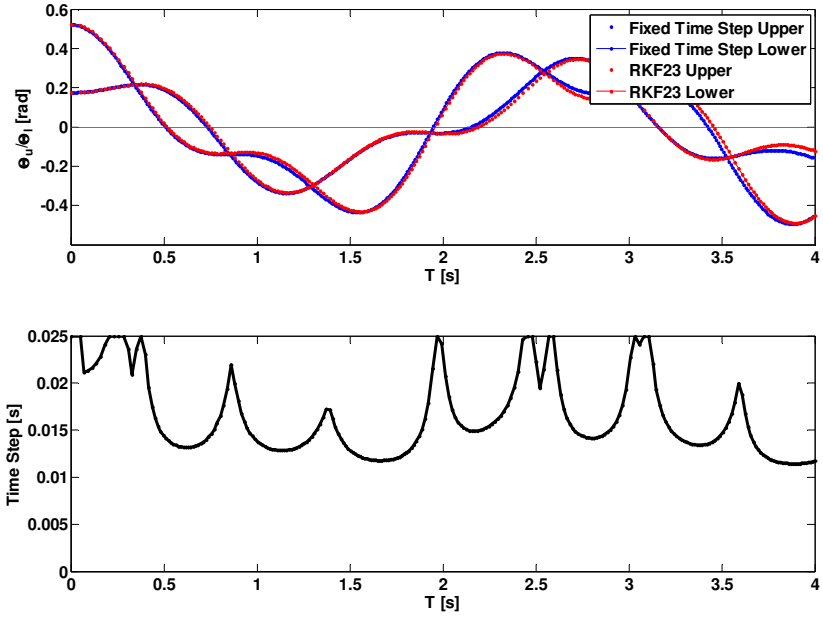

Figure 55. Lowest time step used for both bars, "slow" dynamics

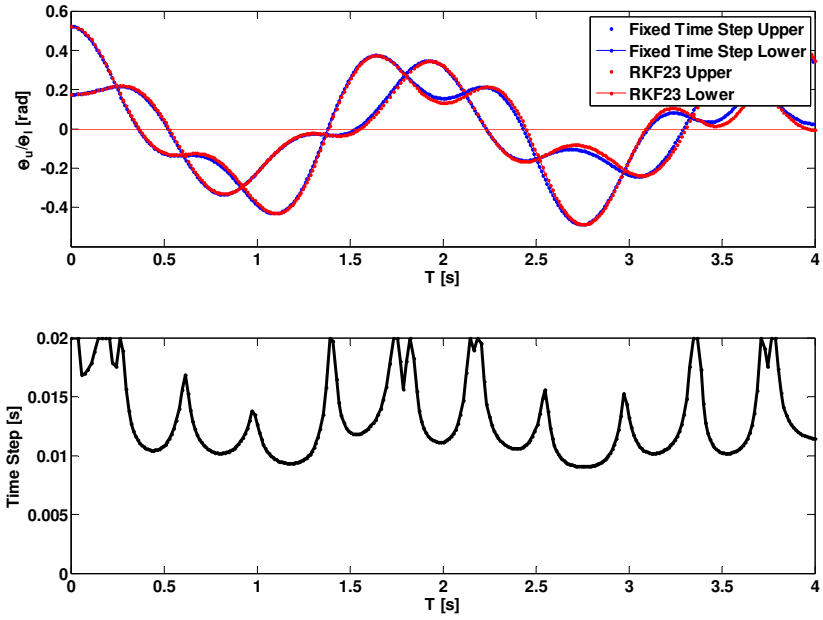

Figure 56. Lowest time step used for both bars, "fast" dynamics

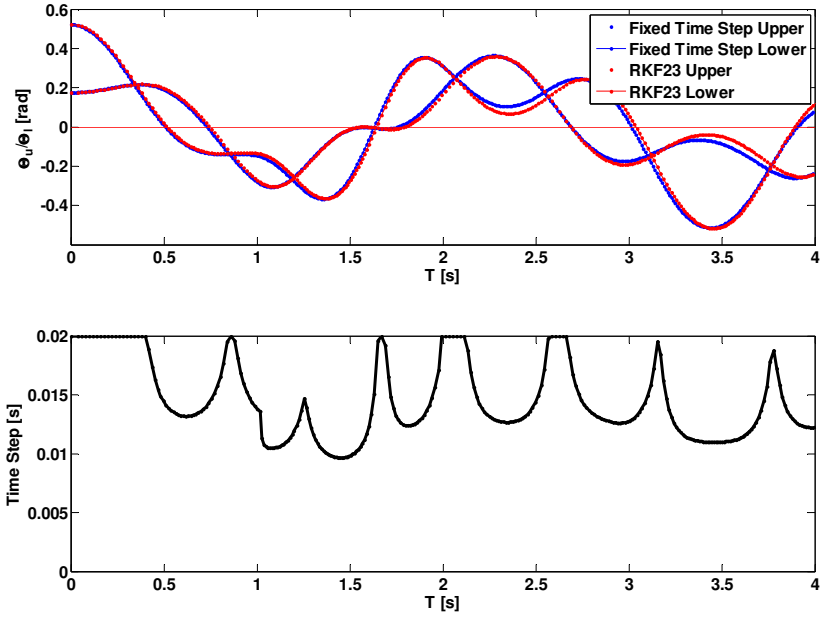

Figure 57. Lowest time step used for both bars, "toggled" dynamics

It can be observed that the algorithm reduces the time step whenever one of the two curves has a high curvature (second derivative). This confirms the applicability of the algorithm for such an application. Table 4 lists the required numbers of function calls.

	Fixed Step	**RKF23**
"Slow" case	1600	1574
"Fast" case	2260	1952
Toggled case	2260	1724

Table 4. Time steps for three cases, with minimum time step used for both pendulum bars

It is clear to see that this simulation setup requires more function calls than the previously discussed setups. This is easy to understand, since the simulation will have to switch to lower time steps (more function calls) whenever *any* of the state variables has a high curvature. Usually, this is the case more often than when only one curve is taken into account. This however, leads the way to another way of setting up the algorithm.

4.2.6 Multi-rate Application of a Numerical Integration Algorithm to Co-Simulation

4.2.6.1 Step 7: Apply RKF23 to all variables, advance each variable independently

In the previous setup, the time step was set according to the fastest dynamic of any of the considered state variables. If all state variables have similar dynamics, then such an approach may be acceptable. If however, different systems have different dynamic behaviors, then the previous application of the algorithm will still result in vast calculation overhead. Such a system (with significantly different dynamic behaviors between the different sub-models) is referred to as stiff system. This is due to the fact that if a system is very slow in comparison to another system within the integrated simulation, then this slow system will not need as many function calls as the fast system to stay within its error boundaries. However, the algorithm presented above does not take this into consideration, but rather applies the smallest time step to all systems, no matter what their respective dynamic behavior is. An extension to the above application of the algorithm can therefore be to consider each sub-model, or rather each state variable, separately, and set the time steps for the slower models differently to that of the faster models. Naturally, this will not be as straightforward as the previous method, because the sub-models will need the information from all the other sub-models for each of their time steps. If a sub-model is slower and needs less time steps, it will still need to submit its state information to the faster models for each of the faster models' time steps. This can be achieved through interpolation of states.

For state interpolation, the algorithm will work as follows: From the starting time point, each sub-model is advance by a time step that is determined for each of the sub-models separately, taking their respective dynamics into account. This will result in different simulation time points for the next time step. If a state variable is far ahead in

simulation time compared to another state variable (a slow dynamic made a larger time step than a fast dynamic), then the slow model is not executed in the next simulation run. Rather, only the faster model (with the lower time step) is executed, utilizing interpolated states from the slower model. The faster model will be executed until it "caught up" with, and potentially "overtook", the slower model. The slower model can then be executed again, with interpolated state information for the faster model. It will bypass the faster model again, upon which it will be stopped. The faster model will then be run again with interpolated data from the slower model until it has yet again "overtaken" the slower model. The algorithm will then execute this "race" until the final simulation time is reached. This is the general approach to the implementation of the algorithm. Figure 58 shows a notional depiction of how this setup should be understood. Figure 59 shows this situation in more detail.

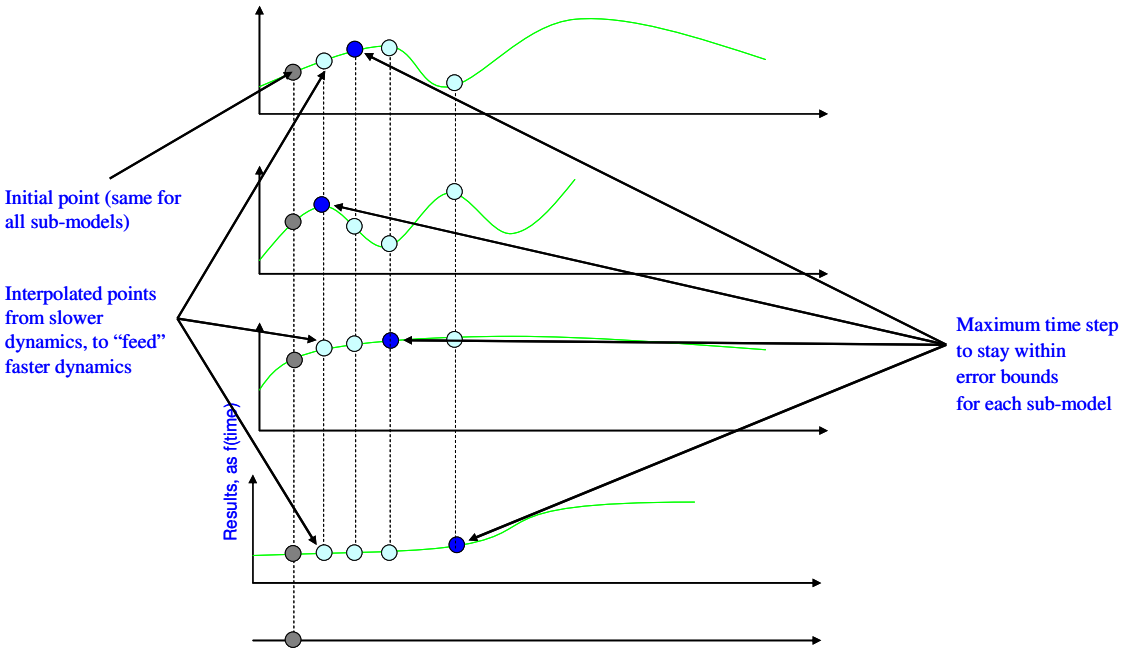

Figure 58. Notional explanation of time step interpolation approach

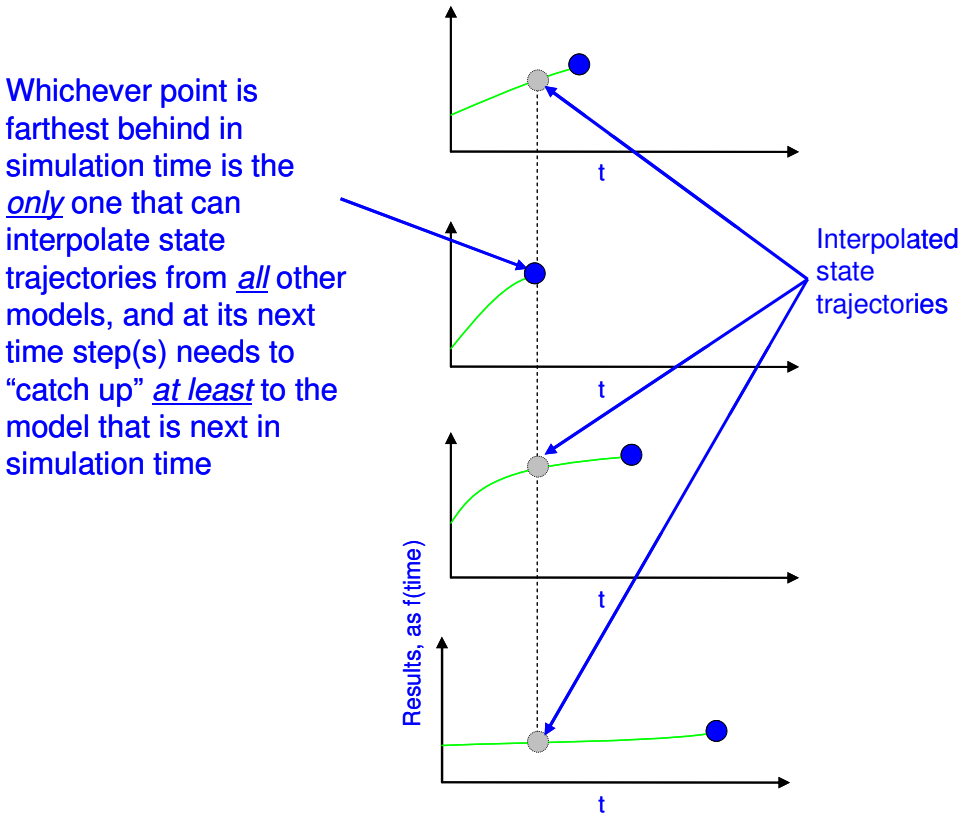

Figure 59. Detailed description of time stepping and updating approach

Figure 60 depicts how this approach is implemented into the actual time stepping scheme. During the stopping time of any sub-model, there is no other model that also stops and delivers data (sub-models with different, dynamically adapted time steps will usually not synchronize unless an integer multiple of time steps is enforced. This however, will complicate the process and run against the notion of adaptive time step for prescribed error, and is therefore not implemented). Therefore, the required system information from other sub-models is interpolated and fed into the halted sub-model, which then can restart for its next simulation run.

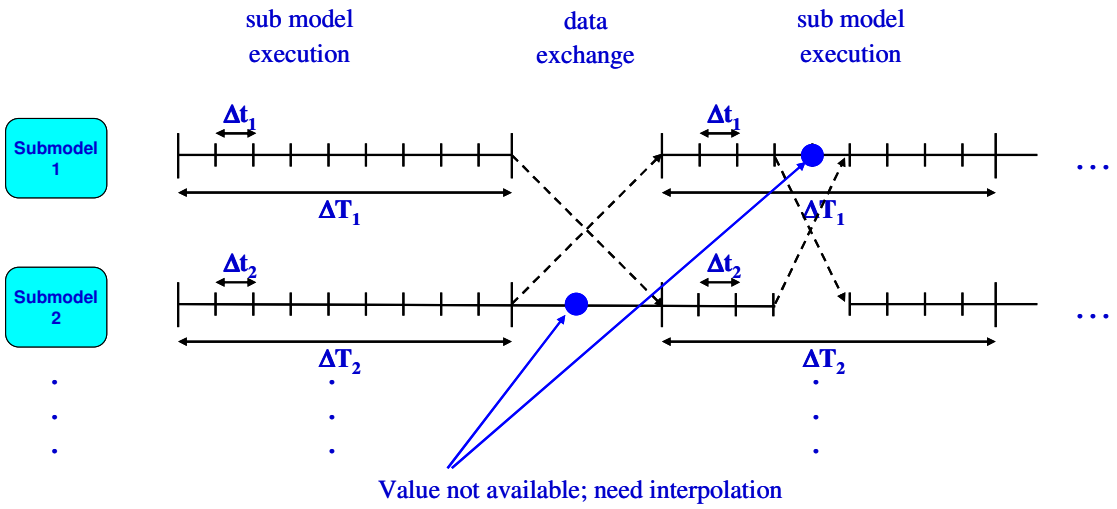

Figure 60. Time step interpolation scheduling

The result of the implementation of this approach to a "mixed" system with a "slow" upper bar and a "fast" lower bar can be seen in Figure 61. The red curve shows the upper "slow" bar and its time step, the blue curve shows the lower "fast" bar and its time step. It can be clearly seen that the two time step histories are significantly different. The upper time step varies much slower than the lower, according to the dynamic behavior of the two respective bars. Also, the minimum time steps for the "fast" case are significantly lower than the minimum time steps for the "slow" system. Therefore, the upper (slower) bar simulation function is called fewer times than the lower (faster) bar simulation function. Where in previous setups, both functions were called equal numbers of time, the slower dynamic of the upper bar is taken into consideration when executing the simulation, resulting in fewer overall function calls. This helps to reduce computational expenses and real world simulation time.

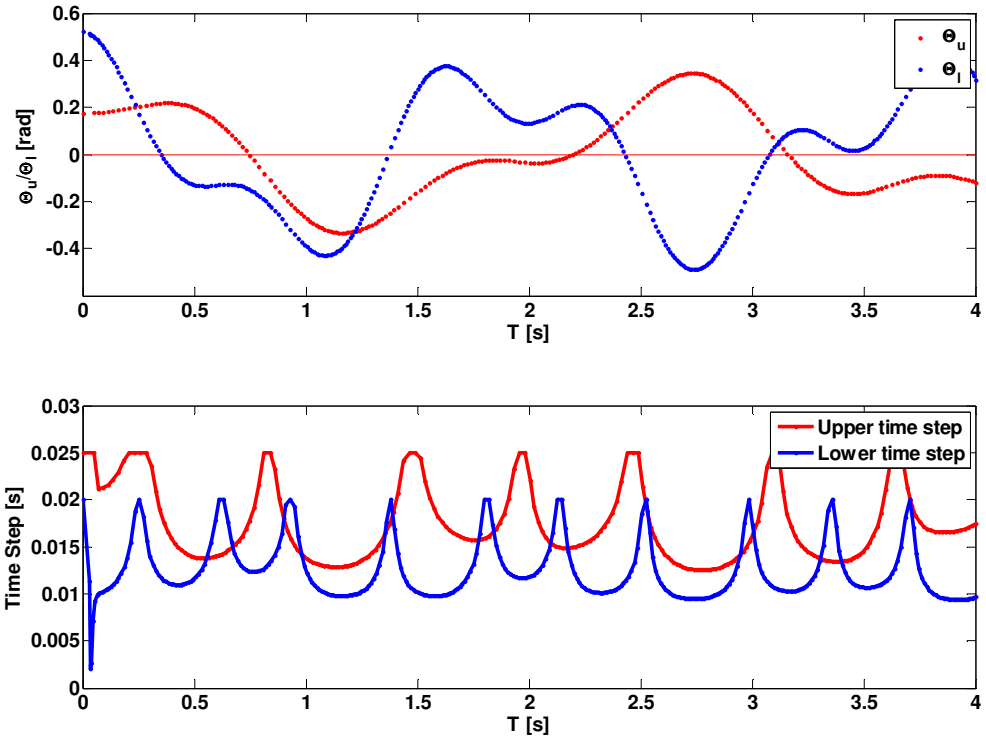

Figure 61. RKF23 time step applied to both bars separately

While in previous cases, both models (upper and lower bar) had the same amount of time steps (function calls), in this new case each bar has its own time step count. The upper "slow", bar required 718 function calls, the lower "fast" bar required 1014 function calls. Table 5 can now be expanded by this case instead of the "toggled" case, to show as follows:

	Fixed Step	**RKF23**
"Slow" case	1600	1574
"Fast" case	2260	1952
"Slow" supper bar, "Fast" lower bar, with separate time steps	n/a	1732

Table 5. Number of function calls required for different cases, incl. separate bars

This shows very well how the separated case lies between a purely "fast" case and a purely "slow" case. Depending on the relations of the different sub-system dynamics, the savings in number of function calls can then be even more significant. Also, this is the correct approach to address the issue of stiff systems, as discussed previously. Stiff systems are such systems where the different dynamic behaviors of the sub-models are significantly different. When this is the case, a distinct adaptive time step must be used for each and every state variable that is exchanged between models. Model execution schedules must be aligned with the time step requirements of those state variables. This may require a more sophisticated scheduler script that can take care of all state variables their assignments to the different sub-models, and the subsequently necessary time stepping of the sub-models. A meta code for this setup is presented in Appendix C.

The code describes how each bar will have to "catch up" with the subsequent other bar until it has overtaken it, upon which the respective other bar will need to "catch up". This is repeated until both bars have reached the final simulation time, given by the user before starting the simulation execution. The required states for that each sub-model requires for the respective other model are acquired through interpolation of the current and previous states. This algorithm would also be generally applicable with multiple models and state variables. In such a case, just like described by the algorithm above, the model that is farthest behind in simulation time would need to be executed with its respective time step until it has caught up with the model that is the farthest ahead. All other lagging models will also have to be executed until they are at least at the same simulation time point as the currently "leading" model. Then the previous "leader" will execute its next step, and the process will repeat until the user prescribed final simulation time point is reached by all sub-models.

4.2.7 The triple pendulum

The double pendulum served as a preliminary "toy problem" to implement the time stepping algorithm. It is a simple dynamic system, and the smallest possible system to be implemented as a co-simulation. The only interactions are between the two pendulum bars. The equations of motions required the small angle assumption, which makes them linear in the angles. It has only two sub-models to take care of during the simulation execution. This greatly simplifies time stepping and data exchange issues, since only two models must be considered and their status with respect to simulation time is easily determined and considered for the time stepping algorithm. While it is a suitable model for first time stepping applications, a more complicated model is needed to make sure that the success of the time stepping algorithm application to the double pendulum was not just a "lucky guess".

The triple pendulum is an extension of the double pendulum. It features one additional degree of freedom in the form of an additional bar and mass, which makes the system more coupled. It also requires more caution with the time stepping algorithm adaptation, and with the data exchange between the models. The additional elements in the models, and the coupling between all three bars, make the equations of motion very involved. These equations are listed in Appendix D. They are not linearized in angles any more, and allow for free variation in bar lengths and masses for each pendulum section. Similar to the double pendulum model before, these equations were programmed into a monolithic Matlab/Simulink model. This model was then split into three sub-models, one for each pendulum, and re-integrated using a Matlab script. Figure 62 shows a notional depiction of a triple pendulum.

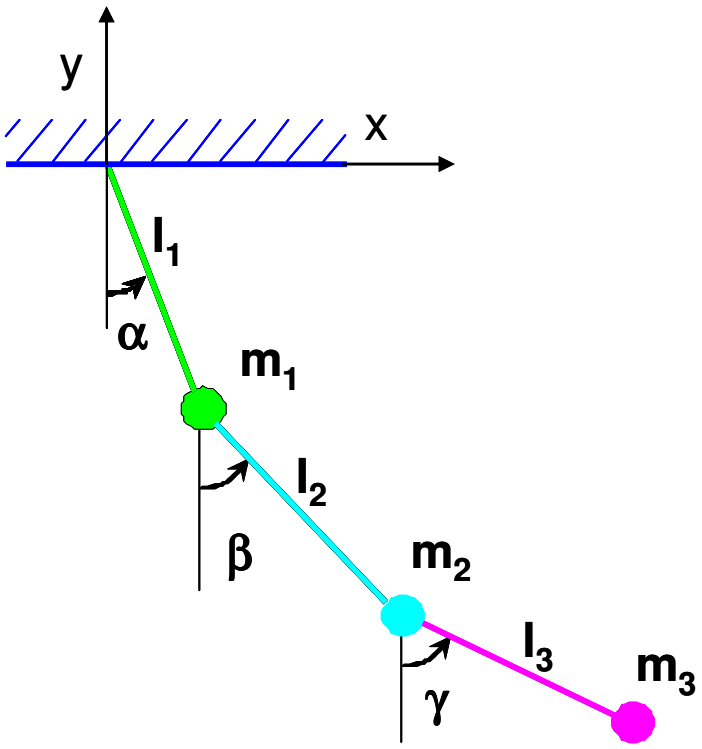

Figure 62. Notional depiction of a triple pendulum model

To test the application of the RKF23 time stepping algorithm to this model, the algorithm was first implemented to the model and the model was run with a "gentle" setup without any extreme properties. The pendulum bars were equally long, and the masses were also equal. Pendulum lengths and masses must be regarded as "unit" lengths and masses, meaning that any unit of length and weight can be applied as long as it is consistent with each other mass and length and the gravitational constant used in the code. For this application, the lengths are in [m] and the masses are in [kg] since the gravitational constant is in [m/s/s], but this is not of importance. The different starting angles contributed a somewhat irregular curve development for the three bars. As before, the number of function calls was recorded for each setup run. Function counts of the monolithic model were compared to the counts for the split model with fixed time step, and for the split model with RKF23 algorithm applied. Figure 63 depicts a sample run, comparing the split model with fixed time step and the RKF time stepped model.

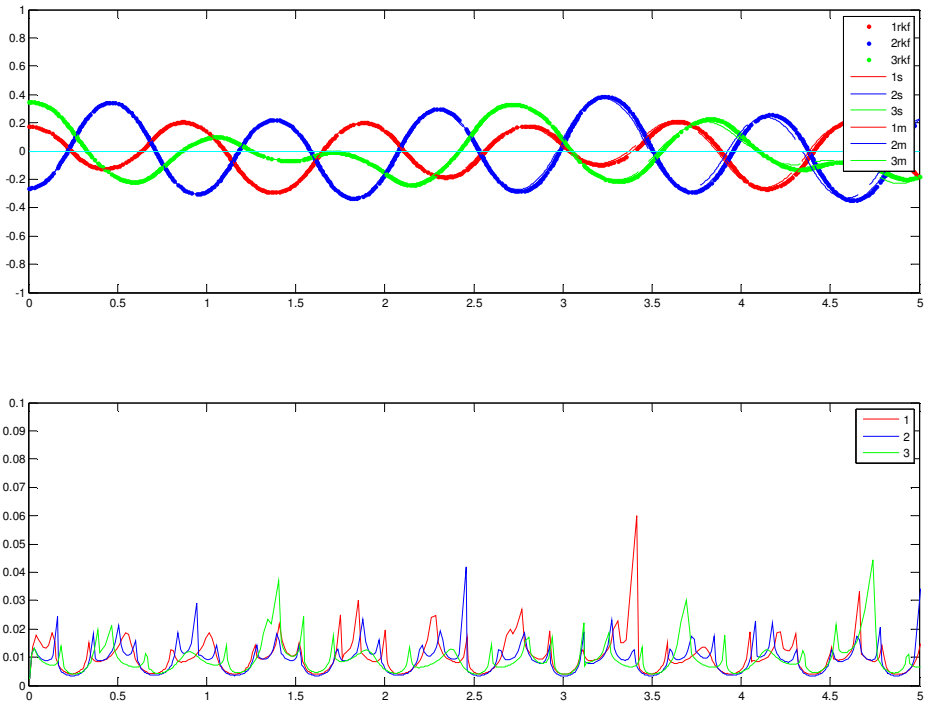

Figure 63. Initial base run of triple pendulum model

The factors that determine the dynamic behavior of the pendulum bars are the mass and the pendulum length. Table 6 shows the system parameters for this setup. Also shown are the starting angles of the three bars, and the RKF algorithm tolerance. As with the double pendulum, the number of function calls is used as the performance criterion to evaluate whether the application of the RKF algorithm actually provides a performance advantage over a fixed step algorithm.

m1	m2	m3	l1	l2	l3	tol.	α_{start} [deg]	β_{start} [deg]	γ_{start} [deg]	split counts	RKF counts
1	1	1	1	1	1	1e-6	10	-15	20	5997	6219

Table 6. System parameters for base case

A note must be made regarding the counting of function calls. As the split model consists of the three sub-models that were "extracted" from the monolithic model and re-integrated, its function count will always be three times as high as the count for the monolithic model. The monolithic model combines all equations in one model, and thus requires only one function call per time step, whereas the split model requires three distinct function calls per time step. Therefore, the monolithic count was left out from further considerations.

When comparing the number of function counts for the simulation, one effectively wants to compare the model efficiency for equal model accuracy. This accuracy decreases with increasing time step size. However, the accuracy should be same for all setups. The split model with fixed time step must have such a time step that its accuracy is always at least as high as the accuracy of the RKF23 time stepped model. Therefore, it must have as its fixed time step that which is the smallest time step that the RKF algorithm has determined over the length of the run. Only then will the fixed step model be as accurate as it needs to be at the location(s) where this smallest time step is required. Hence, for each setup the RKF algorithm was run first in order to determine this smallest necessary comparison time step. This time step was then used for the fixed step split model as its fixed step for all three sub-models.

For the split model with RKF time stepping algorithm applied, the total number of function calls (which is the metric used to compare to the split model with fixed time step) is the sum of function calls required for the three respective pendulum bars. As shown in the double pendulum section, each pendulum bar (= sub-model) has its own time step scheme, and thus has different number of function counts over the run time. While in the split model with fixed time step, this count is the same for all three sub-models, the counts will be different for all three sub-models in the adaptive time step setting. The sum of all the sub-model function counts is used as the comparing

performance metric to the fixed step split model. The justification for number of function counts as a proxy for computational expense of the setup is given elsewhere in this text.

The first simulation results presented above look promising. However, the number of function calls for the RKF implemented algorithm is higher than that of the split model with fixed minimum RKF time step. In such a case, it is obvious that the application of the RKF algorithm is not practicable, as it will increase the cost with no additional gain in error. Why the RKF algorithm may still be useful despite such a situation, is discussed in the section "Additional use of RKF time stepping algorithm" below.

The reason for this disadvantage in function count is due to the fact that the pendulum bar parameters were all equal. Hence, the dynamic behavior of these sub-systems was similar, and the additional expense of the RKF to determine the variable time step was unnecessary. However, one of the main reasons why the RKF algorithm was considered for co-simulation in the first place was the capability of handling different model dynamics between the different sub-models ("stiff model"). Therefore, the easy changeability of system parameters was used to see how the RKF algorithm implementation would handle itself in a situation where the underlying sub-system dynamics were different. In order to test the applied time stepping algorithm for various dynamic behaviors, the pendulum parameters were varied

Multiple such variations were made. A probing was made with respect to the masses. Figure 64 shows the outputs with a mass for pendulum 1 of m1 = 100, with all other parameters equal. Table 7 shows the respective table.

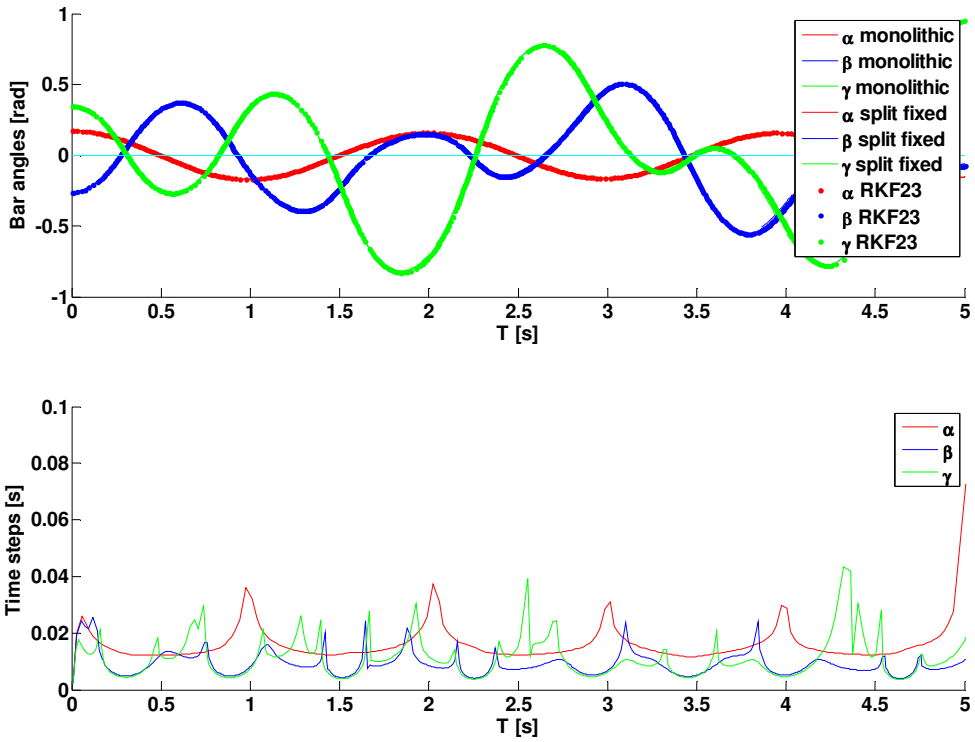

Figure 64. Triple pendulum run with varied mass m1 = 100

m1	m2	m3	l1	l2	l3	tol.	α_{start} [deg]	β_{start} [deg]	γ_{start} [deg]	split counts	RKF counts
100	1	1	1	1	1	1e-6	10	-15	20	5997	4808

Table 7. System parameters for setup with different m1

For this example, it can be observed that the RKF function count is smaller than that of the fixed step split model. This means that in such a setup, the application of the RKF time stepping algorithm is justified to increase the simulation performance. In a next step, the pendulum bar lengths were modified, and the resulting function count was recorded. For both the previous and the current case, the changes were made large in order to investigate how the algorithm behaves with differences in dynamic behaviors

that are several orders of magnitude apart. See Figure 65 for the graph and Table 8 for the data table.

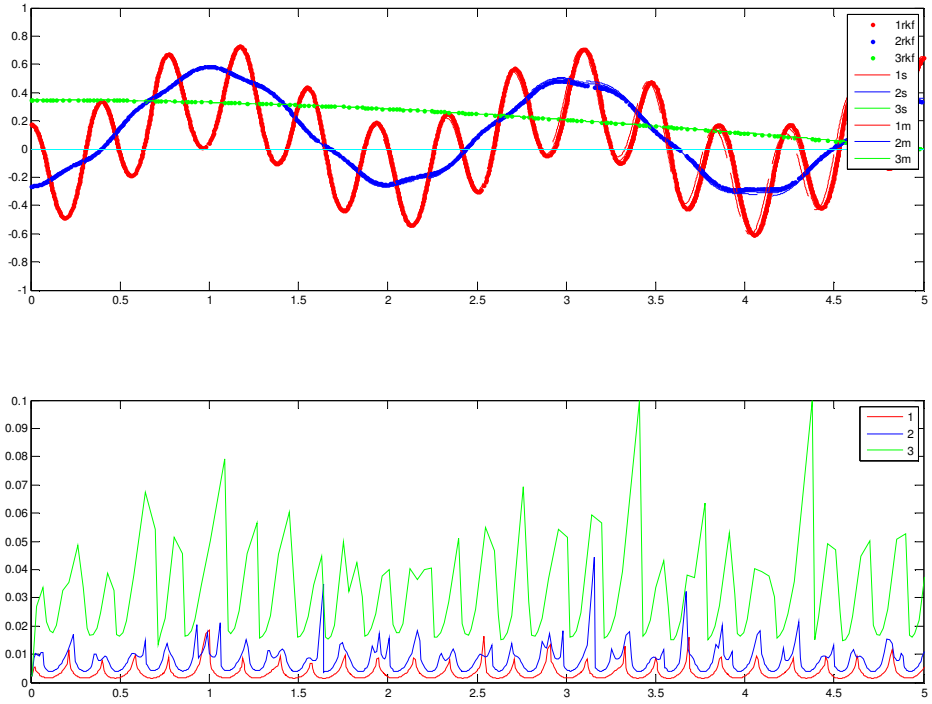

Figure 65. Triple pendulum run with varied bar lengths l1 = 0.1, l3 = 100

m1	m2	m3	l1	l2	l3	tol.	α_{start} [deg]	β_{start} [deg]	γ_{start} [deg]	split counts	RKF counts
1	1	1	0.1	1	100	1e-6	10	-15	20	10002	8800

Table 8. System parameters for bar lengths with orders of magnitude difference

As before, the number of function evaluation calls is significantly smaller using the RKF algorithm.

A side note must be made with respect to the application of the RKF algorithm to a simulation environment. Looking at Figure 65 and the curve for the lowest bar, one can

see that the time stepping points that the RKF algorithm calculated are distributed rather irregular despite the very smooth and extremely slow changing behavior of the curve. In an ideal case, the spacing would be much larger here. However, since the simulation does not always provide such "smooth" transitions between points, especially over wider spacing in time, the RKF algorithm interprets deviations from the actual smooth curve as changes in slope that are larger than they are in the real world. This is due to the way the slopes are calculated within the simulation. In the case where the slopes are available directly from the models and need not be calculated from the system states, the behavior is much more in line with what would be expected for a curve as shown above. This is discussed further in the section "Access to sub-system internals („Grey Box")" further below, where a graph of Figure 65 with slopes directly obtained from the models is used.

In order to get a better fell for how the algorithm behaves in different setup scenarios, a variety of such scenarios were simulated. The main reason is that it needs to be evaluated under which conditions the implementation of a time stepping algorithm is actually preferable to a fixed step method. It could be seen from the few samples above, that there are situations in which the RKF algorithm is actually more expensive than a fixed step approach. Hence, initial tests were run to get a general idea of the situations under which the RKF algorithm is more or less expensive than a fixed step approach.

The following tables present a selection of cases that were run in order to evaluate how the variation of the pendulum masses and bar lengths impact the number of function counts necessary for both the adaptive and fixed step approach. All starting angles were equal for the different cases (10/-15/20 deg. for α_0, β_0, γ_0, respectively. These angles were chosen to provide a more dynamic behavior than the case of similar angles would have given. At the same time, these angles are not too large yet to induce chaotic system behavior). All RKF tolerances were 1E-6. The base case to which all cases are compared is the base case presented above, with l1 = l2 = l3 = 1, m1 = m2 = m3 = 1, and tolerances and starting angles as before. The tables show the different cases that were run, and the

parameters used for each case. Table 9 shows the table for variations in masses, Table 10 shows the table for variations in bar lengths.

Case	m1	m2	m3	l1	l2	l3	fixed counts	counts RKF
1	0.7	1.0	1.0	1.0	1.0	1.0	5,997	6,963
2	0.5	1.0	1.0	1.0	1.0	1.0	6,033	7,748
3	2.0	1.0	1.0	1.0	1.0	1.0	5,997	5,264
4	1.0	0.9	1.0	1.0	1.0	1.0	5,997	6,301
5	1.0	0.8	1.0	1.0	1.0	1.0	5,997	6,295
6	1.0	0.7	1.0	1.0	1.0	1.0	5,997	6,339
7	1.0	2.0	1.0	1.0	1.0	1.0	5,997	6,617
8	1.0	3.0	1.0	1.0	1.0	1.0	5,997	7,248
9	1.0	5.0	1.0	1.0	1.0	1.0	6,726	8,464
10	1.0	1.0	0.9	1.0	1.0	1.0	5,997	5,888
11	1.0	1.0	0.8	1.0	1.0	1.0	5,997	5,690
12	1.0	1.0	10.0	1.0	1.0	1.0	14,790	7,824

Table 9. System parameters for varied masses

Case	m1	m2	m3	l1	l2	l3	fixed counts	counts RKF
1	1.0	1.0	1.0	0.9	1.0	1.0	5,997	6,375
2	1.0	1.0	1.0	0.8	1.0	1.0	5,997	6,622
3	1.0	1.0	1.0	0.7	1.0	1.0	5,997	6,767
4	1.0	1.0	1.0	0.6	1.0	1.0	5,997	7,075
5	1.0	1.0	1.0	0.5	1.0	1.0	5,997	7,259
6	1.0	1.0	1.0	0.1	1.0	1.0	11,214	11,433
7	1.0	1.0	1.0	1.5	1.0	1.0	12,825	9,330
8	1.0	1.0	1.0	1.0	0.9	1.0	5,997	6,658
9	1.0	1.0	1.0	1.0	0.8	1.0	6,156	7,871
10	1.0	1.0	1.0	1.0	0.7	1.0	8,895	9,658
11	1.0	1.0	1.0	1.0	2.0	1.0	5,997	6,190
12	1.0	1.0	1.0	1.0	3.0	1.0	5,997	5,995
13	1.0	1.0	1.0	1.0	5.0	1.0	8,508	6,733
14	1.0	1.0	1.0	1.0	1.0	0.9	5,997	6,357
15	1.0	1.0	1.0	1.0	1.0	0.8	5,997	6,397
16	1.0	1.0	1.0	1.0	1.0	0.5	5,997	7,318
17	1.0	1.0	1.0	1.0	1.0	1.1	5,997	6,181
18	1.0	1.0	1.0	1.0	1.0	2.0	5,997	5,748

Table 10. System parameters for varied bar lengths

To get a better overview over the effects that the different setup configurations have, the data was put in graphical form. Instead of showing the actual function counts, a binary variable was defined that shows only whether the RKF algorithm uses more function calls (0.5) or not (0.0). If the RKF uses fewer calls, the curve is lower and this indicates that using time stepping is advantageous in the respective case and system setup. Figure 66 shows the graph for the mass variations, Figure 67 shows the graph for the bar length variations.

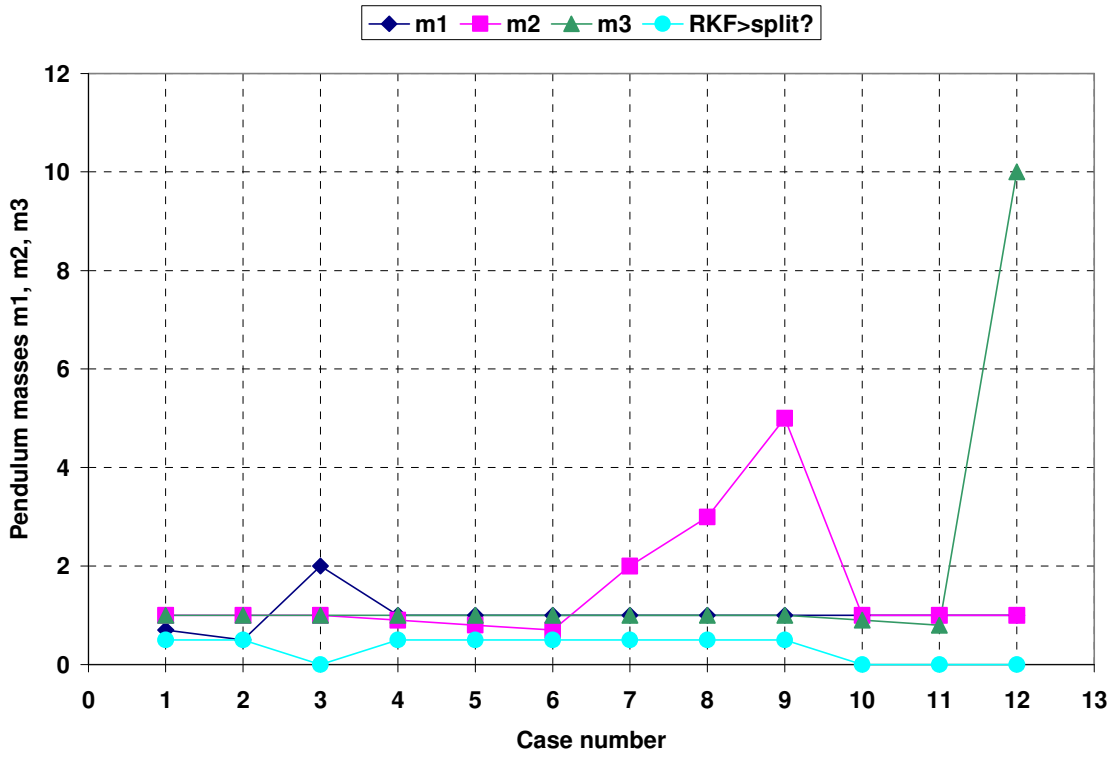

Figure 66. Graph for single mass variations

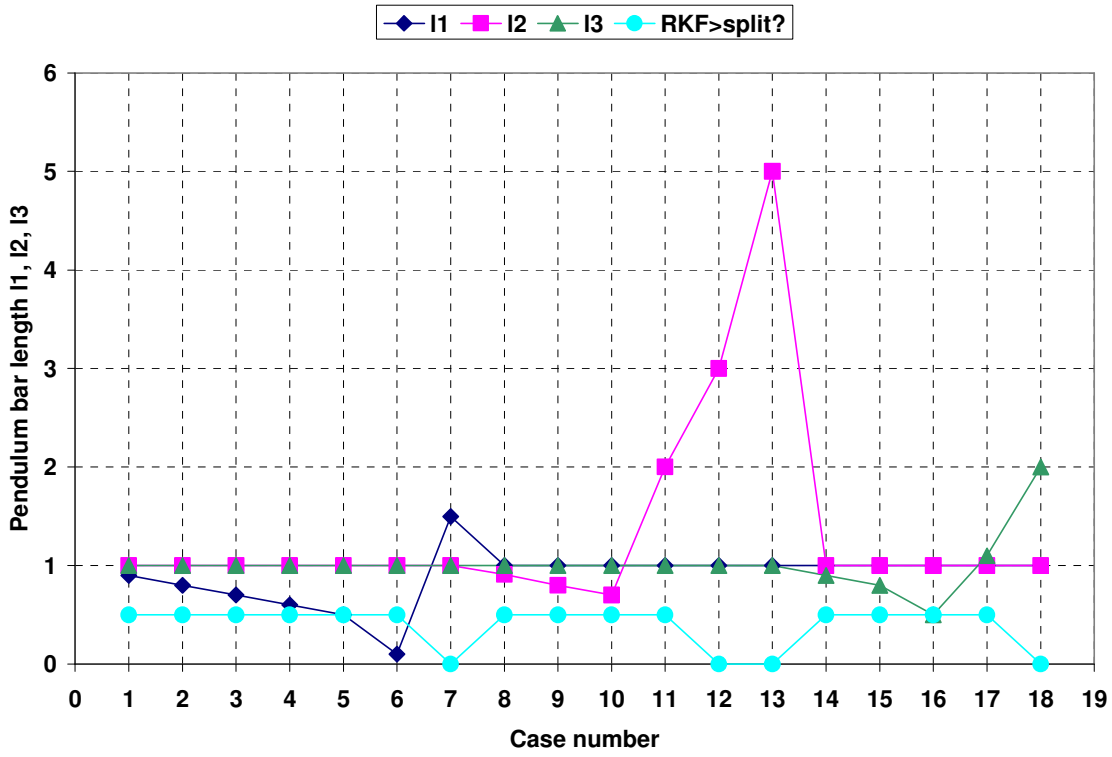

Figure 67. Graph for single length variations

The above tables and graphs are not exhaustive. But they serve as a step into understanding trends that develop with the variation of the model parameters.

The impact of the masses on the decision whether the adaptive time step or the fixed time step is preferable is not equal for all setups. The upper pendulum (m1) puts the adaptive time stepping at an advantage as it increases. Reducing m1 keeps the fixed time step at an advantage. The variation of the mass of the middle bar (m2) does not support use of the adaptive time step at any of the investigated setups. Here, the variations might not have been chosen sufficiently large. For example, m2 could be further increased in order to see whether there is a limiting value from that on the adaptive time step comes at an advantage. The lower bar mass (m3) on the other hand, will set the adaptive time step at an advantage when either increased or decreased from the base case scenario.

The pendulum bar lengths behave more predictable, and also more impactful, than the masses. In general, reducing the length of any one bar will not change the fact that the

adaptive time step is inferior to the fixed step comparison. On the other hand, increasing any one of the pendulum bars will lead to an advantage for the adaptive time step. This however, comes at different length variations. The upper bar (l1) requires only a small increase in length for it to make the application of the adaptive time step advantageous. The middle pendulum bar (l2) requires quite a significant increase before the RKF function count becomes lower than that of the fixed step. The lower bar (l3) also requires a certain minimum increase, not as large as that of l2, but larger than that of l1.

The reason for the adaptive time stepping becoming advantageous over the fixed step approach with changing masses and pendulum bars can be understood as follows. For a single pendulum bar, a longer bar length translates into longer cycle times, and lower oscillation frequencies. This in turn would allow for larger time steps in a simulation, and thus less steps required for a given run time length of the simulation. In a combined setup such as the triple pendulum, this is still valid. However, if only one bar length is reduced, only the function count for this one bar will be reduced. Nevertheless, this represents a reduction in function counts compared to the fixed step model. It must be kept in mind that every single time step count for every sub-model saved in the RKF implementation translates into a saving of three function call evaluations. The impact of pendulum length on system frequency is quite large. Therefore, only small changes may already lead to enough function call savings to equalize or surpass the required function calls for a single bar in a fixed step setup. This is true if not only one but two pendulum bar lengths are changed.

To get a better feel for the general behavior of the algorithm and, more importantly, for the situations in which the application of an adaptive time step will be superior or inferior to that of a fixed step, further investigations were made. The pendulum bar lengths were varied in an orderly fashion in order to find out where and under which conditions an adaptive time step will be advantageous. These investigations were done using the triple pendulum model. It must be noted here that the setup was run

with an RKF error tolerance of 1E-6. This tolerance obviously has an impact on the number of function calls. However, it has been shown in Figure 16 that for each increase in error tolerance by a factor of 10, the number of function calls required to achieve the same error is approximately doubled. This translates approximately into a doubling of the required time steps, as the minimum required time step for achieving the same error will approximately need to be halved for an increase in tolerance by a factor of 10 (e.g., 200 function calls for a tolerance of 1E-2 would need to be 400 for a 1E-3 tolerance). Since the minimum time step directly translates into the required time step or the fixed step model, a pure change in error tolerance will not directly affect the comparisons between fixed and adaptive time steps, and the determination of the scenarios in which each of them is advantageous. An adaptive tolerance level is proposed as a future step for improvement in Chapter 5.

To see the scenarios where an adaptive time step will be superior or inferior to a fixed time step with required minimum value, a full factorial design of experiments was run on the pendulum bars. The pendulum bars were chosen as the variable to be modified due to their larger impact on the results, compared to the masses. Since this investigation is about the dynamic behavior of the model, which can be influenced by changing the masses and/or the bar lengths, investigation results can be similarly applied to varied masses instead of bar lengths. For each of the measurement points a factor was determined that indicates the ratio of function calls for the RKF and the fixed step implementations. Clearly, this ratio is advantageous for the adaptive time step when it is less than unity. The investigations were made at three different pendulum bar lengths, namely 0.1, 1.0, and 10.0 length units. As has been explained, these units must be understood as ratios since their actual values have only a meaning in the context of the actual model. The units of mass and length for the pendulums must be in agreement with the definition of the gravitational constant used in the model. In the current setup, the masses and lengths follow the SI metric system. Figure 68, Figure 69 and Figure 70 show

the case scenarios for the variations of l1 and l2 at constant l3. Note that the axes are log-scaled.

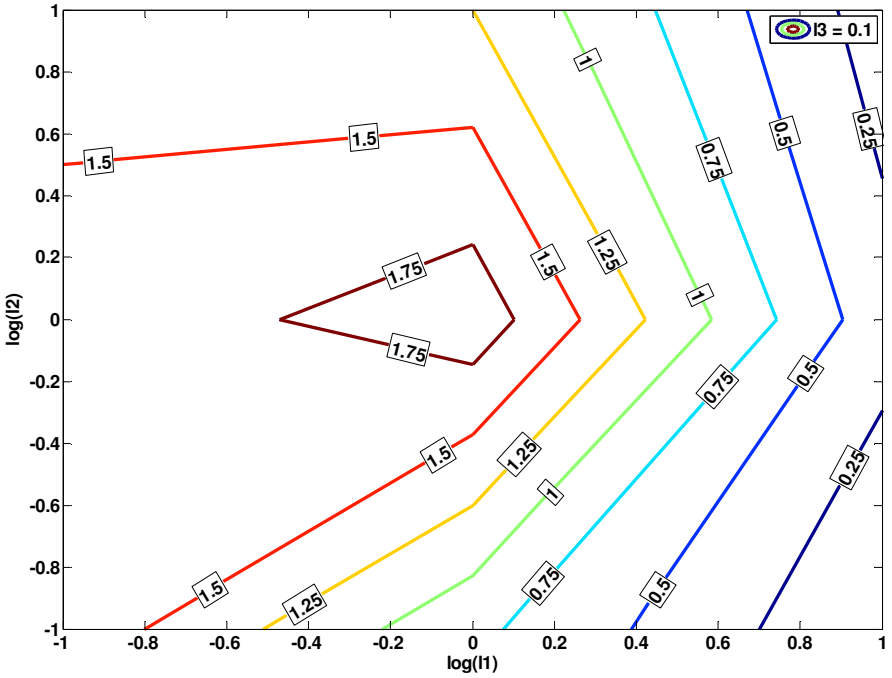

Figure 68. Contour plot for variation of l1 and l2, l3 = 0.1

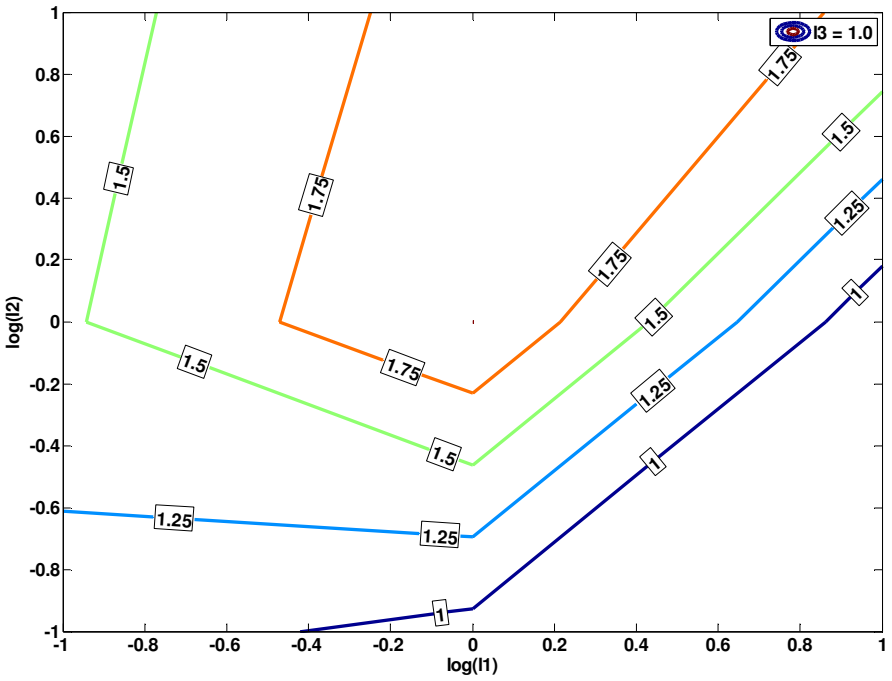

Figure 69. Contour plot for variation of l1 and l2, l3 = 1.0

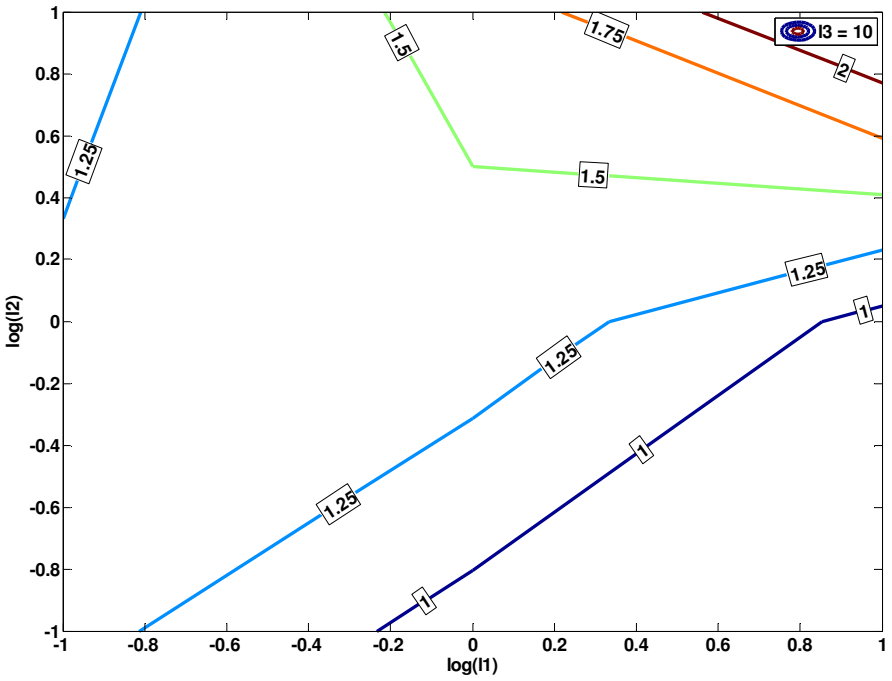

Figure 70. Contour plot for variation of l1 and l2, l3 = 10

The graphs show the iso lines of equal ratio of function calls between adaptive and fixed time step. A higher value means that the adaptive time step requires more function calls than the fixed step implementation, and thus is unfavorable for the adaptive time step. Therefore, the areas with ratios equal or less than unity are of interest, since in these cases the adaptive time step is advantageous for the performance of the simulation.

From the graphs above, several setup configurations can be identified which are representative of distinct areas of advantage or disadvantage of the adaptive time stepping algorithm. These configuration cases are:

1. l1, l2, l3 are about equal length

   This represents a "base case", as shown in the time series plot in Figure 63. In this case, the adaptive time step is usually in a disadvantage. This is because the dynamic behaviors of the three pendulum bars are approximately equal. In such a case, the adaptive time step still evaluates the time steps for all three bars, requiring considerable computational overhead. The fixed step can outperform the adaptive step algorithm in such a case because it will use a fixed step that is equally good for all three bars, without the necessary additional function calls made by the adaptive step algorithm. Hence, in such a setup, a fixed step algorithm is preferable. An exception to this general rule is stated below.

2. l2 and/or l3 ≤ l1

   If l1 is comparatively long compared to l2 and/or l3, then the adaptive time stepping algorithm is usually in an advantage. This is due to the fact that the adaptive algorithm will use small time steps (with high numbers of function calls) only at those bar models that have small lengths and thus fast dynamics. The slower models with longer bars will require less function calls. The fixed step algorithm however, will need to execute all sub-models at the same (small) time step, thus requiring large number of function calls even for the sub-models with

slow dynamics. Therefore, the adaptive time stepping algorithm will usually prove to be more efficient in such a situation. It must be noted that as l2 and/or l3 start approaching l1, there will be a threshold beyond which the dynamics of the sub-models become so similar to each other that the fixed step algorithm will become more efficient again, see Case 1.

3. l1 large/small

   l1 has a special impact onto the behavior of the model and the effect of this onto the performance of the adaptive time step vs. the fixed step implementation. This is due to the fact that l1 is the length of the bar that is connected to the fixed reference system. It therefore has less ability to adapt to forces and moments imposed on it from the movements of bars 2 and 3. On the other hand, it has a larger impact onto the behavior of bars 2 and 3. If l1 is large compared to l2, then the adaptive time step will turn out to be advantageous over the fixed step implementation. This is independent of the length of bar 3, which is not directly coupled to bar 1 and thus has muss less impact onto its behavior.

   When l1 is long compared to l2 and /or l3, the adaptive time step will be advantageous compared to the fixed step. This has been discussed in Case 2. Considering the fact discussed above, that bar 1 is attached to the fixed reference system and exerts more impact onto bars 2 and 3, the slow behavior of bar 1 will to a certain degree impact the dynamics onto bars 2 and 3, effectively slowing them down to a certain degree which then requires less time steps. The exception to this is when both l2 and l3 become similar in length to l1, in which case their dynamics become similar and the fixed step implementation will be advantageous, see discussion Case 1.

   However, when l1 is short compared to l2 and/or l3, then the adaptive time step implementation will become disadvantageous in all the evaluated cases. This is true even when all three bars are of similar length, as the exception mentioned in

Case 1. This behavior must be understood again in the context of the upper pendulum (bar 1) being the one that is fixed to the reference system. If bar 1 is very short, it will show fast dynamics, requiring many function calls to capture the dynamic behavior. Due to its impact onto the lower bars (bars 2 and 3), it will start to modulate the excitation angles of those lower bars according to its own, higher, frequency. Thus, the slower dynamics of the lower bars will become "disrupted" by the impact of the movements of bar 1, and their excitation angles will become modulated by faster dynamics. This then will translate into a higher required number of function calls from the adaptive time stepping algorithm. These modulations can be observed in the following figures, Figure 71 and Figure 72.

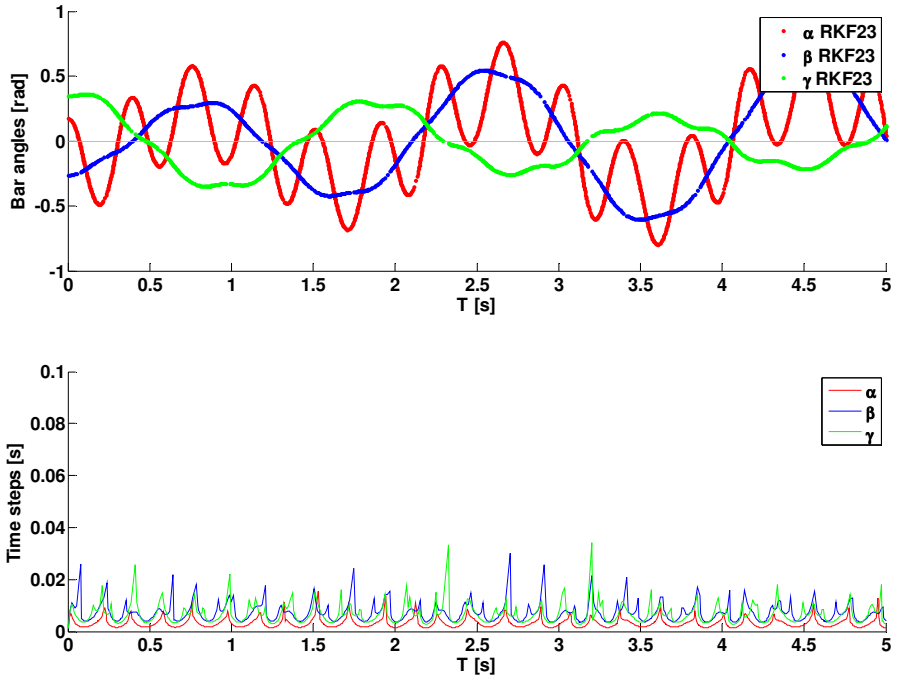

Figure 71. Modulations of bar 1 onto bars 2 and 3

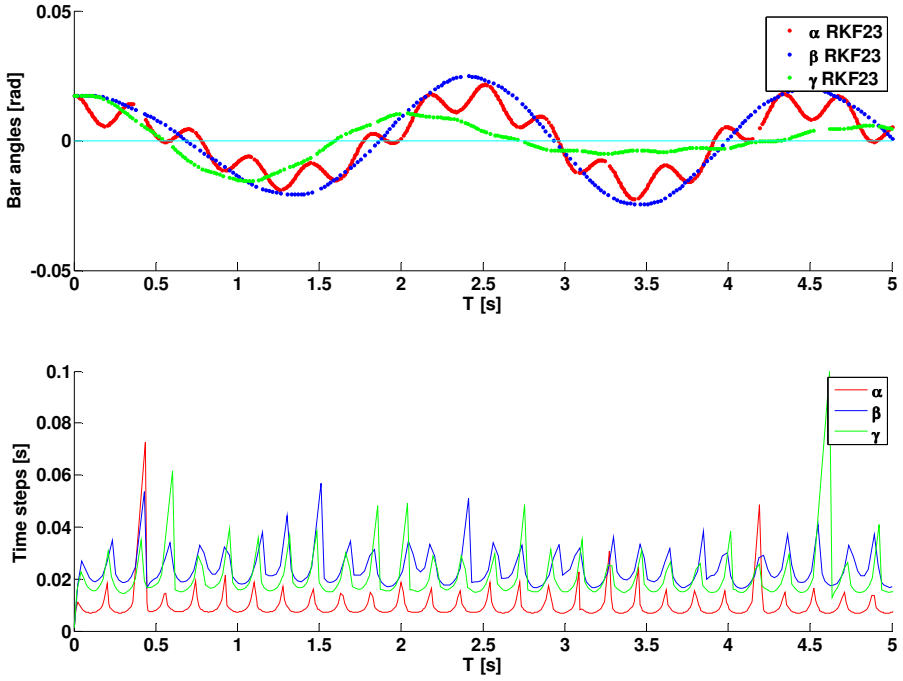

Figure 72. Modulations of bar 1 onto bars 2 and 3, different initial angles

In both figures, bar 1 was at least an order of magnitude less in length than bars 2 and 3. Figure 71 clearly shows how the excitation angles of bars 2 and 3 are modulated by the higher frequency excitations of bar 1. This is also reflected in the time step history in the lower graph of Figure 71. Clearly, the time steps for bars 2 and 3 are modulated following the excitations of bar 1. This is even more evident in the time step history in Figure 72. This simulation run was with equal parameters but smaller initial angle excitations. Clearly, all time steps follow the pattern of modulation from bar 1. This indicates that when a system has a connection to a fixed reference system, and if other sub-models are linked to a model that is the connection to the fixed reference, then there is a chance that the connected models will be affected by the behavior of the model that is connected to the reference system. The closer the sub-models are connected to the "base" model, the larger the impact can be assumed to be.

To demonstrate this, consider the following dynamic system, Figure 73:

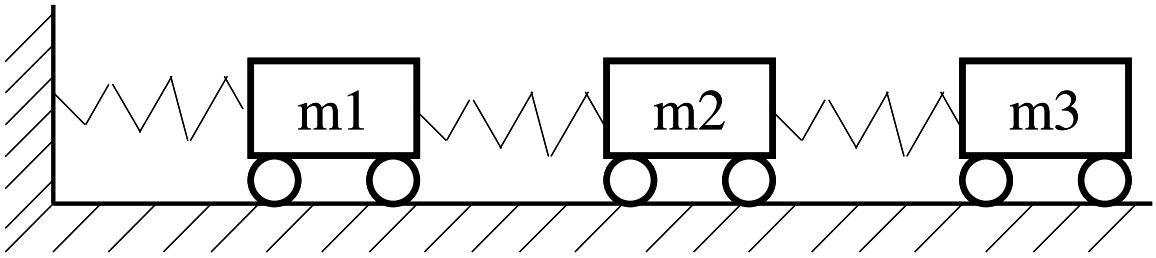

Figure 73. Notional 3-element coupled dynamic system, one sub-system connected to fixed reference system

Such a system can be expected to show a behavior similar to the one described before. Masses m2 and m3 will be impacted by the behavior of mass 1 which is connected to a fixed reference system. If the masses were connected to each other without involving a fixed reference system, such as in Figure 74, then a "symmetrical" behavior can be expected where the dynamics of the model depend solely on the model parameters, and not on a model with a connection to a fixed reference system.

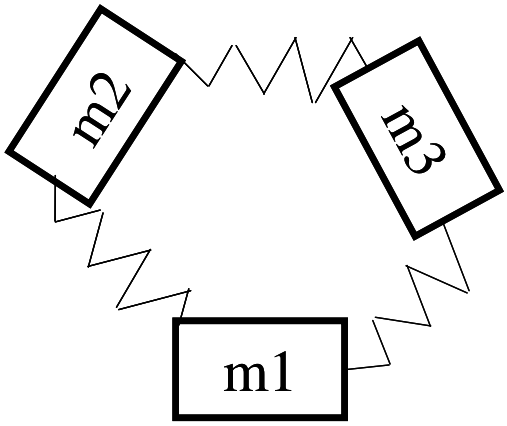

Figure 74. Notional 3-element coupled dynamic system, no connection to fixed reference system

The previous discussion has shown and discussed scenarios in which the adaptive time stepping algorithm can prove itself to be superior to a fixed stepping algorithm. It must be noted though, that these cases were run with the same initial conditions and

masses for each case. For future investigations, it would be interesting to see the impact that changes in these parameters have. The change in lengths of the pendulum bars was justified as having larger impact on the dynamics of the sub-models; variations in the masses should give similar results, as it is about the dynamics of the sub-models more than it is about which parameter leads to the desired model behavior. Nevertheless, these investigations could include further variations, as will be pointed out in the next steps in Chapter 5

4.2.7.1 Behavior with increased state frequency

Another investigation into the applicability of the adaptive time stepping algorithm is to check how it is suitable for increased frequencies. To evaluate this, the model was set up such that the upper two pendulum bars were given extreme lengths while the lowest bar's length was subsequently reduced. Therefore, the upper two bars do not move significantly while the lower bar exerts ever increasing frequencies. Figure 75 shows the result for a setup with l1 = l2 = 1000, l3 = 0.001 (m1 = m2 = m3 = 1, α_0, β_0, γ_0 = 10/-15/20 deg., respectively, tolerance = 1E-6).

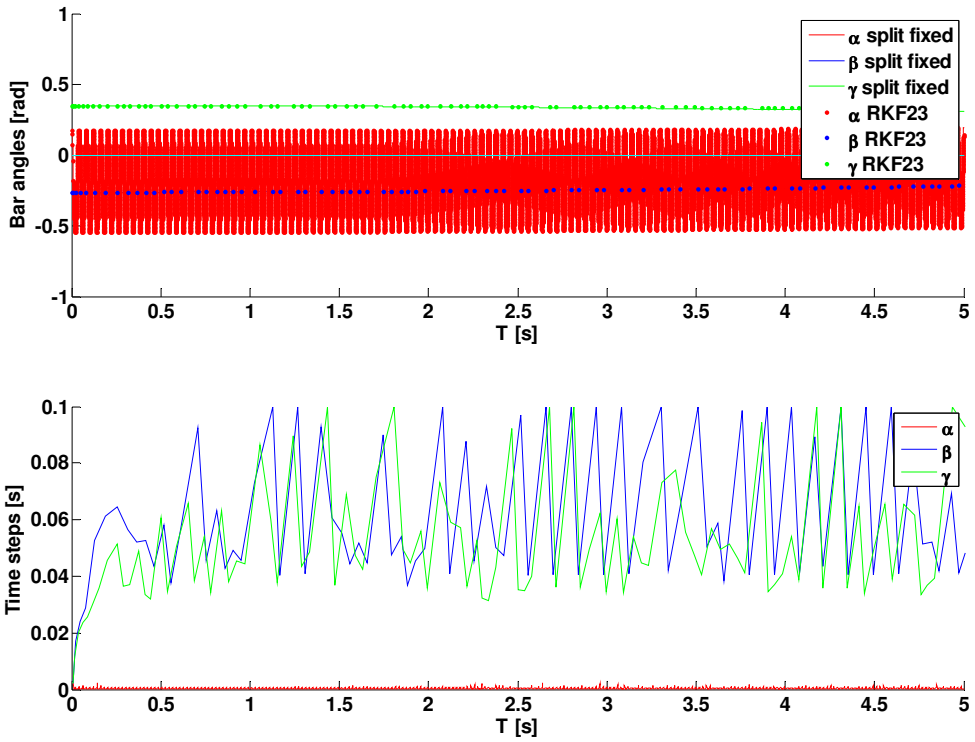

Figure 75. Triple pendulum run with one bar at very high frequency

As can be seen in the chart, the upper two pendulum bars do not contribute to the frequency behavior of the lowest bar. The lowest bar itself swung with a frequency of about 25 Hz. However, what can not be seen in Figure 75 is that the fixed step and adaptive step do not give the same results any more. While the amplitudes and frequencies are still equal, the RKF stepped curve shows a slight frequency shift that is not constant. Figure 76 is an amplification of this phenomenon towards the end of the simulation run.

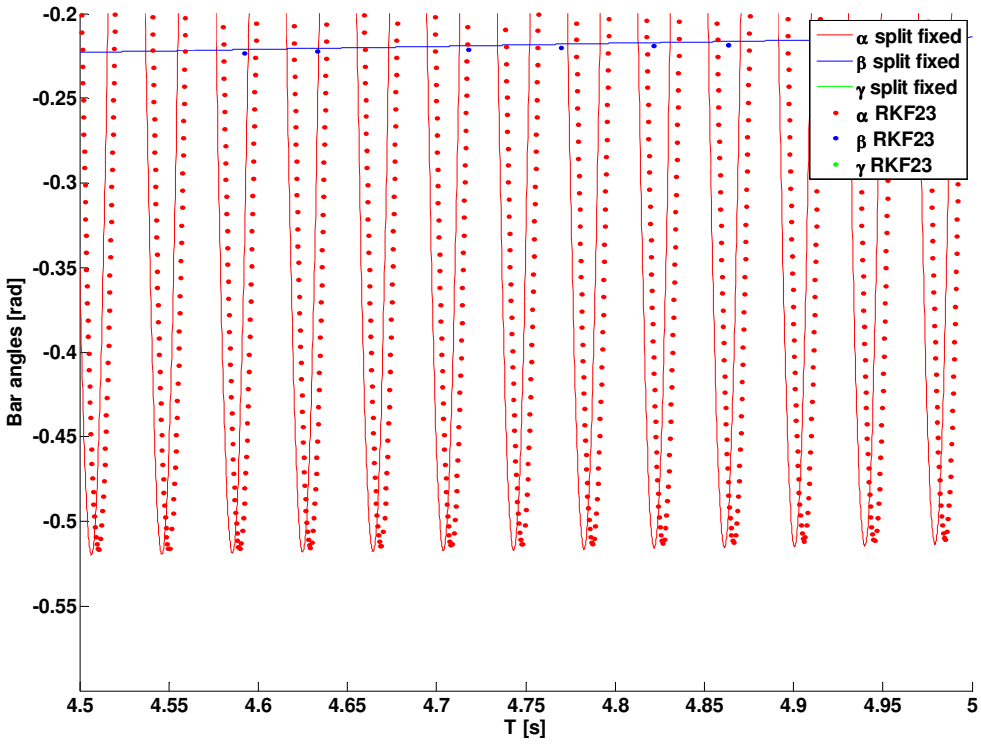

Figure 76. Amplification of high frequency behavior

The observed frequency shifts are not constant, and keep reoccurring during the simulation. This effect is not yet understood. It may be related to modulations due to the movements of the upper two pendulum bars. Since this effect has an impact on the error of the curve, it warrants future investigations.

4.2.8 Additional use of RKF time stepping algorithm

The previous discussions have shown that the application of an adaptive time step to co-simulation of dynamic systems is not always justified. In certain cases, the performance (as measured by function counts) is considerably worse than the application of a fixed time step, even with very small such fixed time steps. Nevertheless, the

adaptive time stepping implementation can be used for other applications, as presented in the following two sub-chapters.

4.2.8.1 Determination of a fixed time step

In general, another use of the algorithm could be the following: It has been shown in the introductory testing case of the triple pendulum that the function count of the implemented RKF23 algorithm was higher than that of the split model with fixed, minimum RKF time step. In such a case, the use of the RKF algorithm is per se not justified, as the split model can get the same result with the same minimum tolerance at less cost. Therefore, if the general dynamic behavior of the system is known to vary only mildly before the simulation, then it could turn out that a constant time step may actually be the better choice. However, if the internals and the actual dynamic behavior of the sub-models is not known (the "Black Box" assumption), then the question still is how to determine a fixed time step that can be used for the simulation. It has been mentioned before, that techniques such as system identification tools are not appropriate for all kinds of systems because they can only fulfill their task if it is certain that the underlying system is linear. It is hard if not impossible to determine this for any given system over the whole range of its input state variables. Hence, other methods need to be used, some of which are described below. Nevertheless, if the general range of state inputs is know beforehand, the simulation engineer can use the RKF23 algorithm to determine a fixed time step of his system by applying the algorithm to a sample run. If this sample run covers the fastest dynamic behavior that can possibly occur during any simulation situation, and if the model dynamics are known to not change too much over the execution of the simulation, then the fixed time step obtained through the RKF23 (the lowest that occurred during the sample run) can be used as a fixed time step for the simulation. In this application, it is ensured that the fixed step will then always deliver at least the required error tolerance, as it was given for the RKF algorithm during the testing

run. This is also true for the earlier cases where it has been shown that the RKF algorithm performed worse than a fixed step algorithm that had the minimum time step of the RKF algorithm applied. The knowledge of this minimum time step was only given through the previous evaluation runs of the RKF algorithm. Under such model setup conditions, the approach of using the RKF algorithm to find a fixed time step will be applicable and preferable.

4.2.8.2 Handling of varying dynamics

One main justification of using an adaptive time stepping algorithm was to be able to account for varying dynamic behaviors of the sub-models during simulation execution. Therefore, as with the double pendulum, the dynamic behavior of the simulation was changed during the run time in order to evaluate the stability of the algorithm to such changes. Figure 77 depicts the time series plot for this configuration, Table 11 gives the function call results and the model parameters.

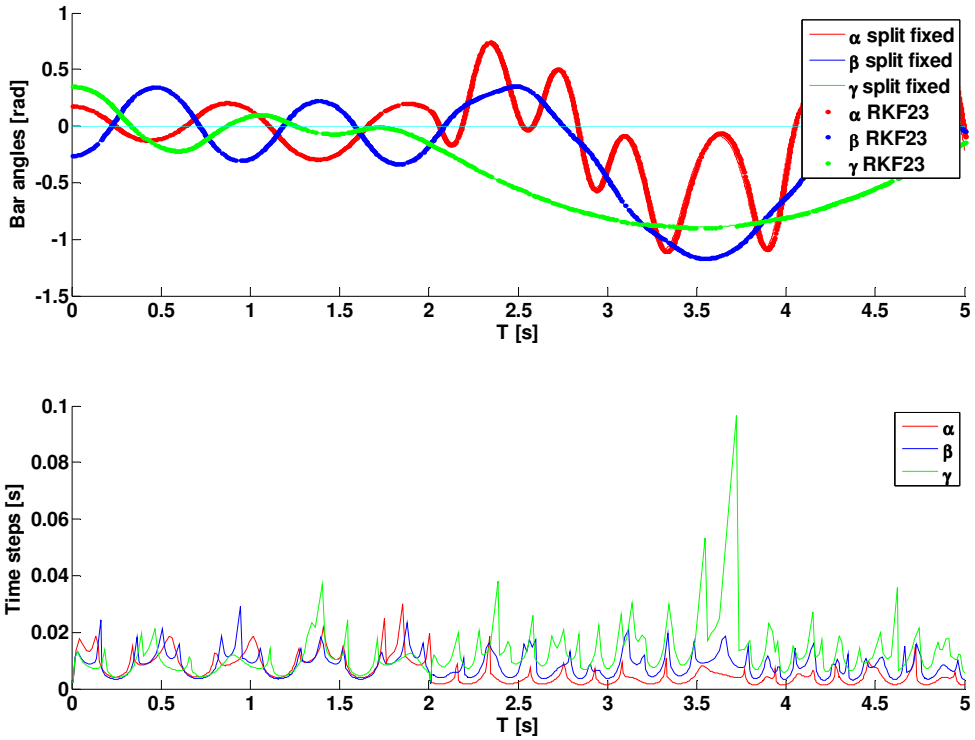

Figure 77. Triple pendulum run with changing sub-system dynamics

m1	m2	m3	l1	l2	l3	tol.	α_{start} [deg]	β_{start} [deg]	γ_{start} [deg]	split counts	RKF counts
1	1	1	1	1	1	1e-6	10	-15	20	11727	8215
1	1	1	0.1	1	10		n/a	n/a	n/a		

Table 11. System parameter variations during simulation execution run

The upper row of parameters is the starting condition. At simulation time T = 2 seconds these parameters were changed to the ones shown in the lower row. It can be observed that the algorithm adapts to the changing dynamic behavior by reducing the time steps of bars with the reduced lengths. The advantage of the adaptive algorithm over the fixed time stepping is immediately obvious. The fixed step would, as discussed before, have to operate with a minimum time step that covers all eventualities regarding changing model dynamics. If the dynamics are slow, the fixed step will nevertheless work with a small time step, thus requiring many more time steps than would be necessary during the phases of slow dynamics. In such scenarios, the adaptive time stepping implementation will be more efficient. This will be true the more such cases occur during the run time, and the larger the changes in dynamics of the sub-models are.

4.3 Further Issues And Considerations

The previous demonstration of the algorithm and its implementation into dynamic system co-simulations has shown that a time stepping algorithm from numerical integration of ODEs can be employed to find a suitable time step for the co-simulation. By executing sub-models on an “as needed” basis, the amount of function calls, and thus computational expense and real world time, could be significantly reduced under certain conditions. The implementation of the algorithm involved different steps, some of which require a bit more consideration in future implementations of the algorithm. The following list points

out some of the issues and considerations to be investigated. This list is by no means complete, but it should help to identify points where the algorithm may still have opportunities for improvement or extension. Other issues and considerations can be imagined and will no doubt come up if this research is taken further in the future.

4.3.1 Trajectory interpolation

The final implementation of the algorithm, namely the execution of the sub-model that is "furthest behind" in simulation time, requires the interpolation of the state trajectories of the remaining sub-models as inputs for the "slowest" sub-model to be executed with current overall system conditions. In the examples chosen (double and triple pendulum), the state trajectories behave smoothly. No jumps or sudden increases or decreases in state trajectories are to be expected. Therefore, the trajectory interpolation can be performed using a simple polynomial curve fitting between the state trajectory points of each of the remaining models. This however, may not provide acceptable inputs any more once the underlying trajectory does not behave smoothly any more. Steep ascents or descents in states may cause the polynomial to over-swing and hence the interpolated values may not properly represent the underlying curve any more. Under such circumstances the interpolated values will not be representative of the actual system states, and deliver wrong information for the currently executable sub-system.

To clarify this, consider Figure 78 where the underlying state trajectory is smooth, with moderate change in slope (second derivative is low). A polynomial curve of third order was fitted to the four node points, and represents this curve accurately. Deviations between the node points are minimal, and acceptable for a state trajectory interpolation.

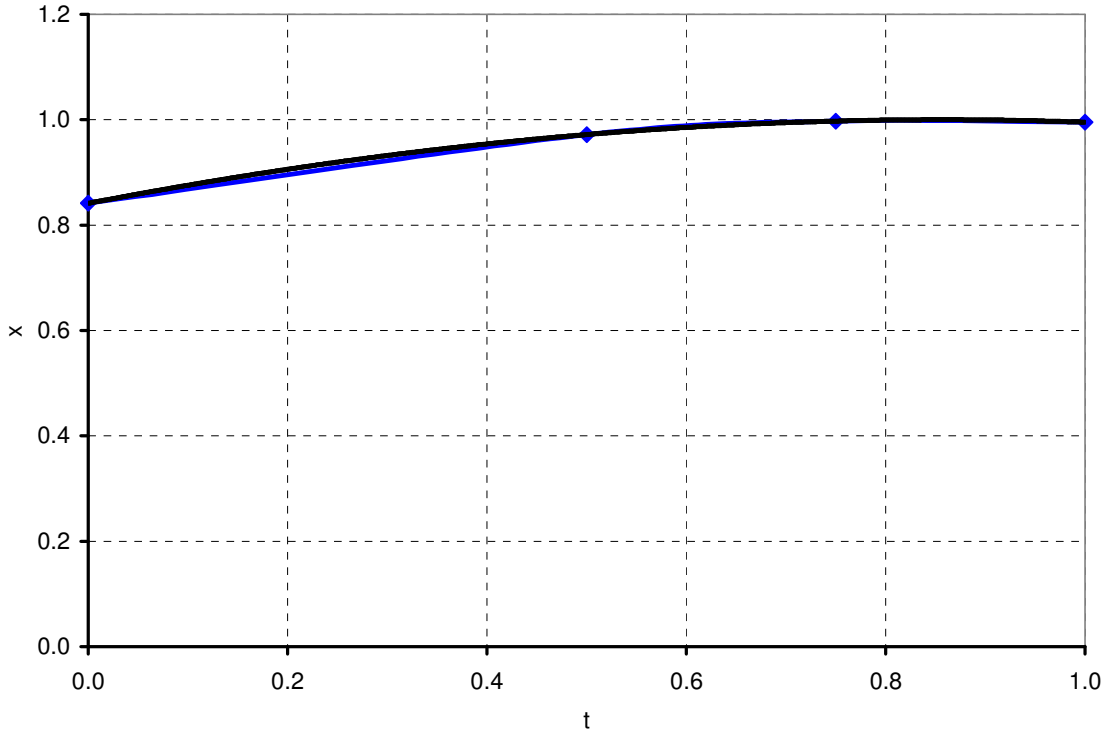

Figure 78. Smooth trajectory example

If however, the underlying curve does not behave smoothly as before, the result will be quite different. Consider Figure 79 where the last point drops sharply away from the previous three points. In this case, the fitted polynomial does not accurately represent the underlying trajectory. While the polynomial approximation does still go through all the node points, it does not accurately represent the curve in the sections between the node points. Using such a curve would result in high deviations of the state trajectory inputs for the currently executed sub-model in a co-simulation. Depending on the robustness of the model and the setup, this might result in unstable model behavior and wrong model outputs.

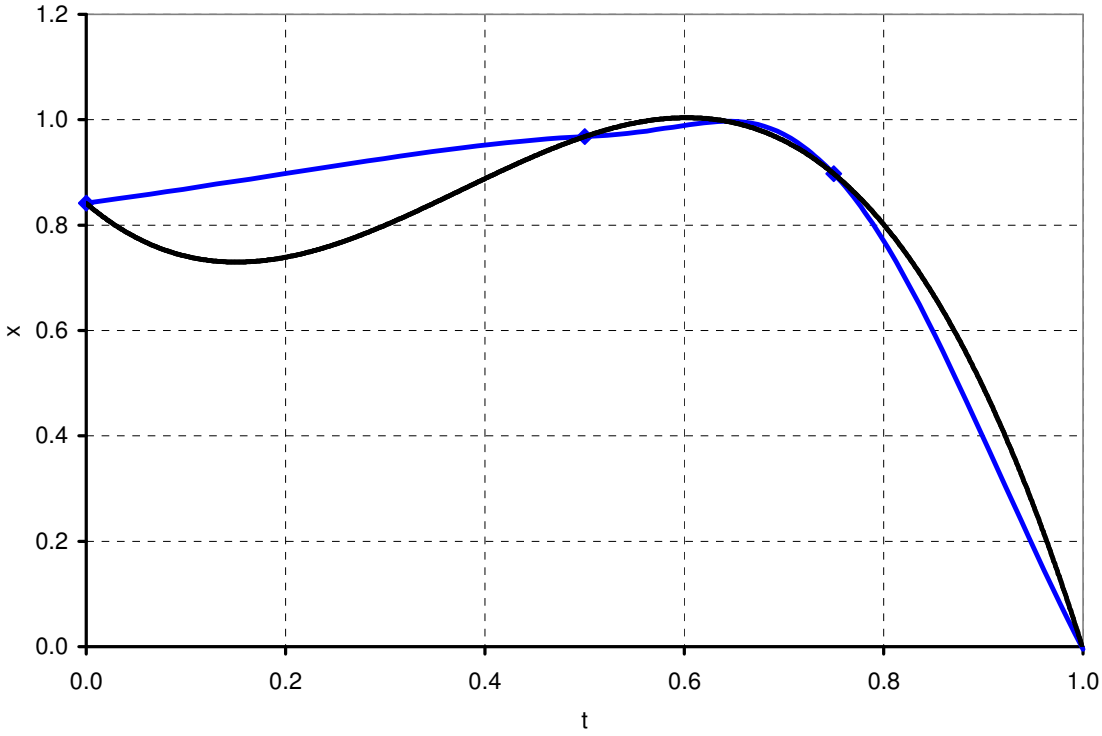

Figure 79. Trajectory curve example with sharp drop

In order to achieve better interpretations of the underlying system trajectory behavior, different interpolation and function approximation routines have been employed and tested for their suitability for this problem. The over-swing behavior of polynomials under such conditions is well known, and multiple methods have been developed to overcome such behavior and allow for better curve interpolation under non-smooth conditions of the underlying curve.

A Matlab script was programmed to display and compare multiple such algorithms. The investigated algorithms include multiple interpolation methods implemented in Matlab itself, and additional sources of algorithms that represent implementations of other algorithms not readily available from Matlab. The routines provided by Matlab are:

- Polynomial fit (POLYFIT function)

- Spline interpolation (SPLINE function, piecewise interpolation)
- Another Spline implementation (INTERP1 function with ‘spline’ parameter)
- Piecewise cubic Hermite interpolation (INTERP1 function with ‘pchip’ parameter)
- Other Spline algorithm implementations (INTERP1 function with ‘cubic’ and ‘v5cubic’ parameters, respectively)

Additional external function implementations are:

- A Lagrange interpolation algorithm implementation
- A Radial Basis Function (RBF) algorithm implementation
- A Bezier interpolation function as implemented in Microsoft Excel (and used in Figure 78 and Figure 79 as the underlying state trajectory curve)
- A modified Bezier interpolation function implementation

Figure 80 and Figure 81 compare the outputs of the different interpolation algorithms to each other, using the two sample curves discussed above.

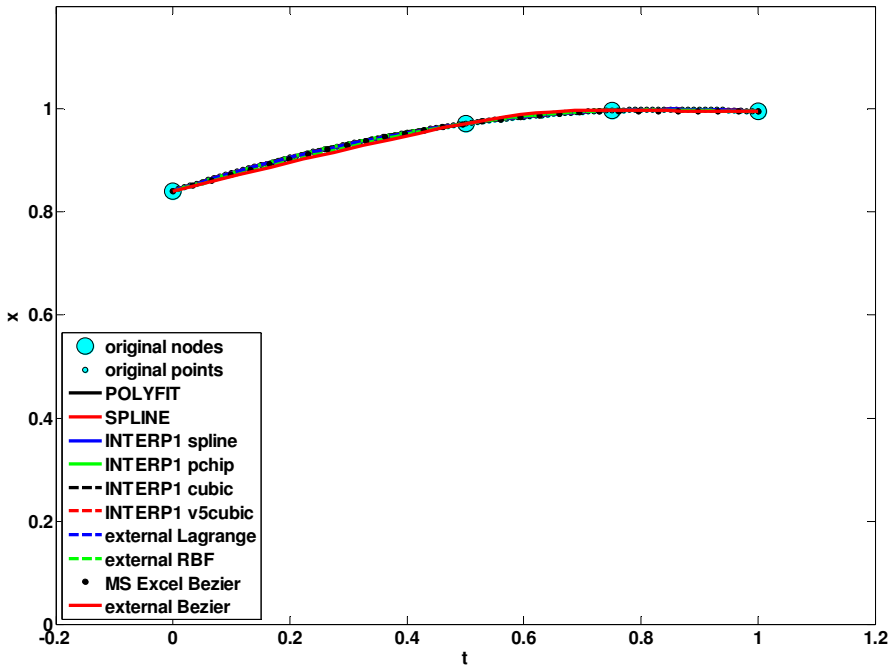

Figure 80. Smooth trajectory with different interpolation algorithms

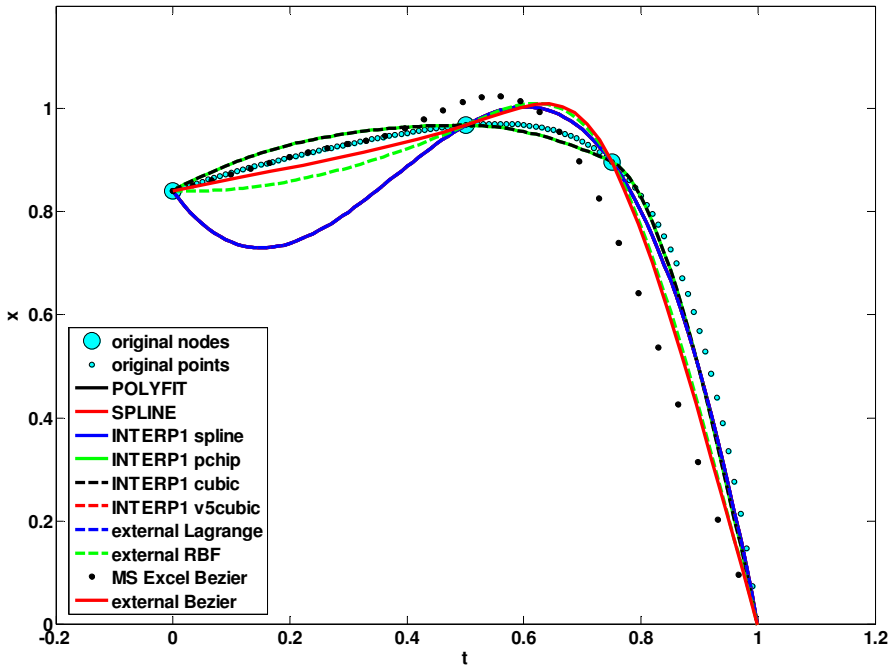

Figure 81. Trajectory curve with sharp drop, different interpolation algorithms

For the smooth curve, not surprisingly all algorithms behave similarly well. The non-smooth curve is more of a challenge. Here, the normal polynomial fit performs by far the worst. All other methods perform more or less reasonable. While several of the methods show a slight over-swing between the second and the third point, generally the behavior is at least as good as the polynomial fit. In the next section, there will be a discussion with respect to the derivative of the curve at the node points, and the requirements of the approximated function with respect to these derivatives. In the meantime, a decision must be made as to which algorithm may be preferable with respect to the state trajectory implementation. In order to help make a more informed decision, a second underlying sample trajectory curve was chosen with a similarly steep change in state. The same interpolation methods were applied again, and evaluated against each other with respect to suitability for trajectory interpolation. See Figure 82 for an illustration of the curve.

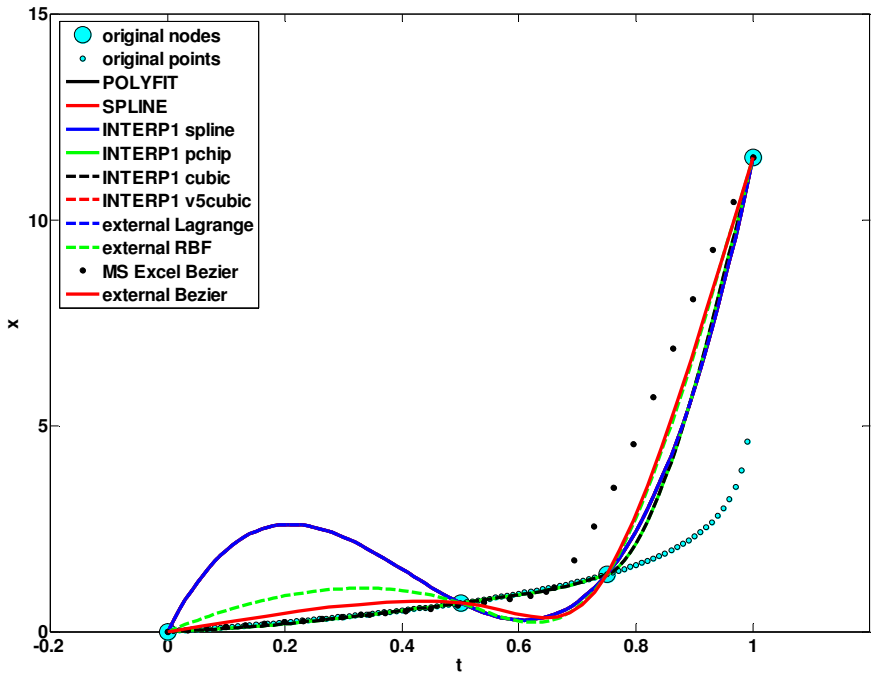

Figure 82. Trajectory curve with sharp drop, mathematical equation

This function is the inverse of a function presented in Süli and Mayers (2003). The equation for the curve used here is given as y = -ln(-x+1) * epsilon, with epsilon=1. It can be seen that this function has a very steep change in slope between c.a. 0.8 and 1.0. In fact, the point at 1.0 lies in infinity, and the last node point considered in this example is at 0.99999.

It can be readily observed that most of the algorithms fail to produce suitable outcomes in this example. Some of the curves that showed promising outcomes in the previous example do not live up to the expectations in this example. The only two algorithms that can follow this second example (which is of course a quite drastic one) are the piecewise cubic Hermite interpolation (INTERP1 function with 'pchip' parameter) and the Bezier interpolation function from an external source. The piecewise cubic Hermite interpolation however, is slightly worse in the previous example. Therefore, the Bezier implementation is assumed to best fit for the role as the interpolation function for the state trajectories. It seems to be suited best for both smooth and non-smooth state trajectory behaviors, and is deemed best suited to be able to cope with drastic changes of the state trajectories (high rate of change of the slope = high second derivative). The reference and source code for the Bezier algorithm as implemented here are shown in Appendix E.

4.3.2 Determination of the slopes

The description of the underlying mechanism to determine the time step in a co-simulation has pointed out that the slope of the state trajectory is required to perform the applied RKF algorithm within the co-simulation environment. One of the main points for this thesis text was that these state trajectories are not readily available from within the sub-models. Hence, they will need to be approximated by the information available about

the states through the state trajectories. As for the state trajectory interpolation, a simple third order polynomial fit was used with the current state, the states at the half-step and full-step future steps, and the state at the most recent previous time step as node points. The previous discussion about the state trajectory interpolation has shown that a polynomial is not always suitable for this task. This comes immediately evident in Figure 79, where the polynomial trend line is clearly not representative of the slopes on the node points. While the slopes at points 3 and 4 may be acceptable, the slope at point 2 is clearly too high, and the slope at point 1 even has a different sign. Hence, this approach should not be used as a general case. Therefore, the previously introduced functions would need to be considered with respect to their ability to provide accurate derivative information based on, and at the location of, the underlying node points.

The test cases are the functions presented in Figure 81 and Figure 82, because they are clearly defined functions with equally clearly defined derivatives. Therefore, they allow for direct comparison of the approximated derivatives with the actual derivative values. Figure 80 is not considered because all algorithms give good results in the smooth case.

It can be seen from Figure 81 and Figure 82 that some of the functions will not be suitable for the derivative approximation in the non-smooth case due to their large overshoots and deviations. For Figure 81, the following functions have proven to be acceptable approximations to the derivatives at the node points:

- Piecewise cubic Hermite interpolation (INTERP1 function with ‘pchip’ parameter)
- Other Spline algorithm implementations (INTERP1 function with ‘cubic’ and ‘v5cubic’ parameters, respectively)
- A Radial Basis Function (RBF) algorithm implementation
- A modified Bezier interpolation function implementation

For the function in Figure 82, the following functions have proven to be acceptable approximations to the derivatives at the node points:

- Piecewise cubic Hermite interpolation (INTERP1 function with 'pchip' parameter)
- Other Spline algorithm implementations (INTERP1 function with 'cubic' and 'v5cubic' parameters, respectively)

Hence, for derivative approximation, the Matlab functions for piecewise cubic Hermite, and the Spline interpolation methods are the most promising. However, while these algorithms are better than polynomials in determining local derivatives, they are sill not perfect in all respects. More advanced methods for derivative approximation could be tested in a future step, such as fuzzy algorithms, Kalman filters, scattered data approximation approaches, etc.

4.3.3 Computational overhead

The previous two issues discussed are a necessary side product of the RKF23 algorithm implemented into a co-simulation. However, they still represent an additional computational overhead expense, which needs to be taken into account when determining the usefulness of the RKF23 implementation. Therefore, the additional computational expense must somehow be quantified and compared to the actual expense for executing the sub-models in the co-simulation setup. In order to do this, a Matlab script was written which compares the execution speed of a sub-model with that of some of the interpolation algorithms. The time a computer requires to execute either a sub-model or an interpolation algorithm can be used as a proxy to estimate the computational effort.

The sub-model used was one of the sub-models for the double pendulum. It represents the simplest dynamic system possible, a single ODE. In order to avoid model-internal time stepping, the setup was such that the solution to the model was to be found in one single step. This prevents the solver from performing expensive time stepping internally, thus increasing the computational expense. Thus, this setup is the computationally cheapest model.

The sub-model setup was compared to some of the previously identified interpolation and derivative algorithms, namely the polynomial fit of third order, the external Bezier algorithm, and the Matlab INTERP1 function with 'pchip', 'cubic', and 'v5cubic' parameters, respectively. The input data to these algorithms was a previously generated set of random values to exclude their generation time from the run time measurements, and equal for all functions.

1,000 runs were done for each respective setup. All runs were done on computers with Microsoft Windows XP operating systems. However, since this operating system is "non-deterministic" (one does not know what other threads run in the background and how they slow down the computer and its execution speed for the simulation), the setup was executed on several different computers and multiple times. This of course lead to different real world time requirements for the executions, but what is the actually interesting part is the relation of how much time was for each interpolation algorithm compared to the simulation of an actual dynamic system. (On a side note, this was also the reason why real world execution time was not given as a metric for comparing the double and triple pendulum models) Therefore, the data was collected and the ratios of required times with respect to the sub-model execution were calculated. Since it is distributed data, there is a mean and a variance:

	polynomial	ext. Bezier	'pchip'	'cubic'	'v5cubic'
Mean	41.1	48.4	38.8	40.6	20.6

Variance	3.90	2.93	14.28	0.22	0.80

Table 12. Ratios of execution times simulation model / interpolation algorithm

Table 12 shows the relation of execution times of the interpolation algorithms with respect to the single-step sub-model execution. External Bezier is roughly 50 times faster, the Matlab INTERP1 function with the 'v5cubic' parameter is still roughly 20 times faster. This shows that the computational overhead due to the additional calculations for trajectory interpolation and derivative approximation is negligible, also with respect to real world execution time. Furthermore, the calculation of these additional functions has no impact on systems that are distributed over different computers. In such a setup, the data exchange time and amount is critical due to the bandwidth and connection times required for networked computers. Trajectory interpolation and derivative approximation do not add to required bandwidth in networked co-simulation setups.

In the evaluation of the results for the co-simulation setups with the double and triple pendulums, function count was given as the main metric to evaluate the performance of the RKF23 time stepping algorithm compared to a fixed step algorithm. The discussion about the additional computational overhead due to miscellaneous function calls for trajectory interpolation and derivative approximation has shown that execution the respective sub-models is always much more expensive than any other calculations that need to be done for every time step. One must keep in mind that the comparisons above were done with the simplest possible dynamic model with the least computational expense necessary: a single ODE model, solved in one step. Any other dynamic sub-model will likely be more expensive to run. Therefore, every function call is expensive, and a reduction in function calls (while controlling the error) is therefore the main performance goal for the time stepping algorithm. The use of function calls within the co-simulation as a performance metric is therefore justified.

A last note on the issue of computational overhead is that even if the expenses for trajectory interpolation and derivative approximation were higher in comparison to the sub-model expenses, it would still be required in order to implement the time stepping algorithm properly. The RKF23 algorithm will give the required minimum time step for a given system state. This step can not be increased in order to save computational expense, unless a compromise in error is accepted. Therefore, despite additional computational expense, this approach will still provide better performance and controlled error.

4.3.4 Access to sub-system internals („Grey Box“)

The basic idea of this thesis text was to be able to find a time stepping algorithm that can be sued in a co-simulation environment where there is no knowledge about the internal dynamics and/or mathematics of the sub-models. All information to and from the sub-models was assumed to be in the form of system state variables. This was referred to as „Black Box“. However, this approach could be loosened somewhat by assuming that the sub-models do not only return state variables but also their derivatives. This has two distinct effects: First, it will render the necessity to estimate the derivatives from the states obsolete. This means that the computational overhead discussed before will reduce to the state trajectory interpolation only. The second effect is then, that the error that is introduced into the system through the derivative approximation is non-existent in the new setup. Hence, the results for the co-simulation state values will be more accurate, resulting in an improved interpretation of the state values and a better time step adaptation. Figure 83 is a reproduction of Figure 65 with the same parameters, but derivatives used directly from the Simulink sub-models, not approximated from the state values.

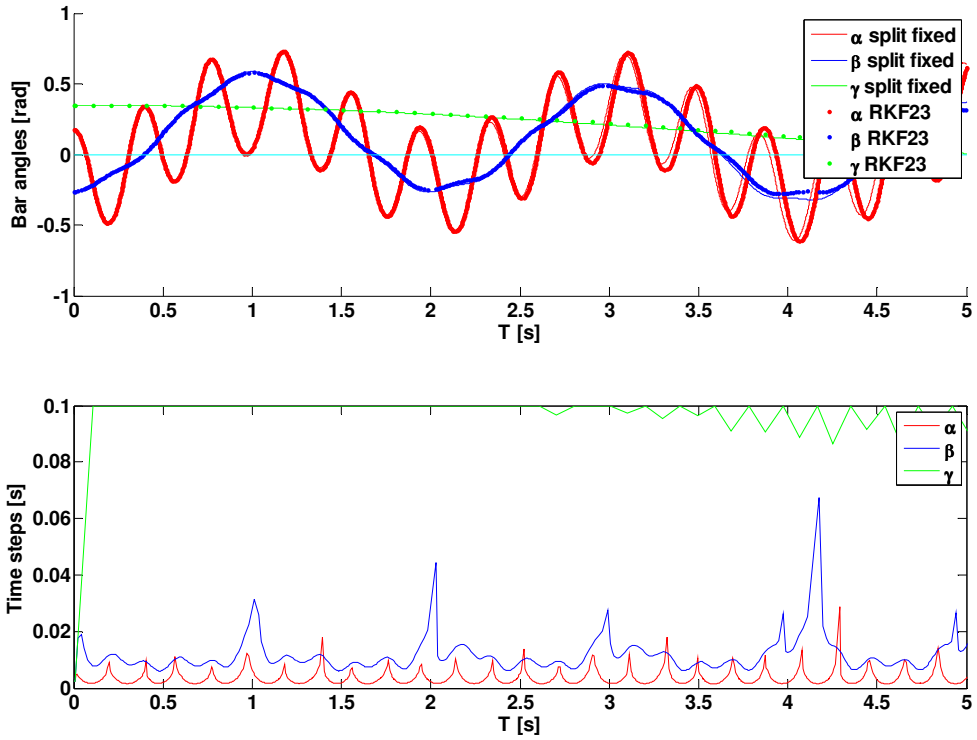

Figure 83. Implementing state derivatives directly from model (not from interpolation and approximation)

m1	m2	m3	l1	l2	l3	tol.	α_{start} [deg]	β_{start} [deg]	γ_{start} [deg]	split counts	RKF counts
1	1	1	0.1	1	100	1e-6	10	-15	20	10422	7391

Table 13. System parameters for direct derivative implementation

Figure 83 and Table 13 show that the function count could yet again be reduced further through the readily available derivatives. Observing the result, one can also see that the time step for the slowest model is at its prescribed maximum over almost the entire run time length. Increasing this time step threshold to beyond the current value would further reduce the amount of time steps necessary for the slowest model, and thus further reduce to overall time step count, making this approach even more effective. Also for the slowest model, the time step irregularities observed in the setup with

approximated state derivatives are greatly reduced due to the much higher accuracy of the derivatives and the non-existent dependency of the derivatives on the behavior of the state values.

Another consideration of "Grey Box" models might be the ability to have access to sub-steps that are calculated within the sub-models during their respective execution. In terms of the time step descriptions in Chapter 2, this would refer to the local time step Δt within the sub-models. While the access to such model-internal information would render the interpolation of states between the models obsolete, it would not have a direct impact on function counts and execution speed. This can be understood if one imagines the time steps within the models available. Having this information readily available would mean that it can be fed between the models at any global time step ΔT in order to provide state information between the models. However, in order to fit to the global time step for each sub-model, the respective sub-models will still need to be stopped at their respective time step in order to synchronize their states with the other sub-models. The state values of the other sub-models is readily available without state interpolation, but the number of function calls will not be reduced because the time step for the sub-models will not change due to the available information. However, the solver for such a setup would need to be much more complicated because it would require that the solver accesses the current sub-model's information during run time and makes it readily available for any other sub-model that may require the data. This will require a coupled solver for such an approach, which comes along with much greater programming efforts due to the increased workload for sub-model synchronization and data exchange. The payoff of such an approach with respect to the main performance metric "function call count" will likely not justify this effort.

4.3.5 Hybrid simulation

In this thesis text, the simulation was of the *continuous time* type. While the real world continuous time has to be discretized in order for a digital computer to be able to handle the timing, the underlying nature of this simulation type is still a continuous time assumption. However, a second type of simulation has steadily gained momentum and importance, namely that of *discrete event* simulation. It has been touched on in Chapter 2. Discrete event simulation does not assume a continuous time approach. Rather, it only states times within the simulation when certain events are to be taking place. For this, each sub-model has its own internal clock running, and the trigger for the event is based on a prescribed behavior of the model. Discrete event simulation is widely employed in warehousing, and is based on queuing theory.

One major application of discrete event simulation is the modeling and simulation of failure times of technical systems. For example, given a certain failure rate, times of system or component failures are computed using e.g. exponential or Weibull probability distributions. When the failure time has arrived during the simulation, the failure event will be triggered. Failure, safety and reliability models have become an integral part of system design, modeling and simulation due to their critical importance in systems engineering. Hence, it is not to come across such models, and to fulfill the necessity to include them into a co-simulation setup. This however, will pose problems with respect to time stepping. It is clear to see that a discrete event simulation will not follow a time step that has been prescribed through a time stepping algorithm onto the co-simulation sub-models. As mentioned, the discrete event sub-model will have its own internal clock, and trigger the event based on its prescribed rules, irrespective of the current time step currently within the continuous time model realm. Figure 84 depicts this scenario in a continuous time environment.

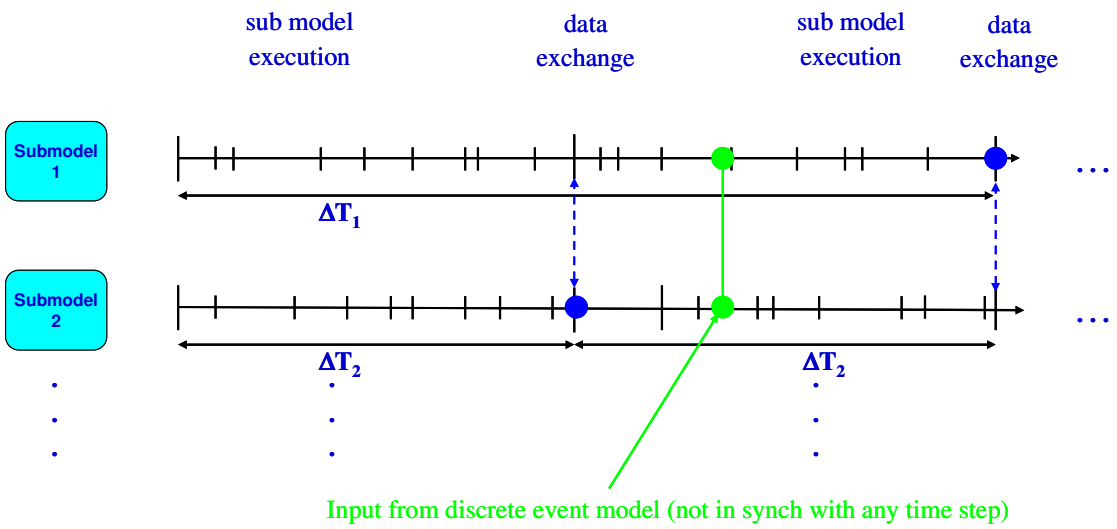

Figure 84. Time line scenario for discrete event simulation in continuous time model setup

Based on the works of Fujimoto (2000) and Law and Kelton (2000), Lee at al. (2001) describe different timing mechanisms for both continuous time and mixed model setups. Purely continuous time models can be updated using either a synchronous fixed time interval (all models are updated at the same fixed time step), a synchronous variable time interval (all models are updated at the same variable time step; this was the first testing step for the double pendulum in this thesis text), or an optimistic time warp method, which involves prediction of events occurring and rolling back the simulation accordingly. The last method presented may turn out to be quite inefficient. A fourth method is asynchronous timing with re-synchronization. Asynchronous timing is what has been applied in the last step for the pendulum models in this thesis text. Each sub-model advances with its own time step as necessary. However, if a discrete even model is included, its inputs must be taken into account at the time step it requires. Such a discrete time event will most likely not coincide with any of the calculated time steps for the continuous time models. Hence, if at any time within the simulation a discrete event has been triggered, the simulation needs to stop and synchronize the simulation and all sub-models to the time step where the discrete event occurred. This likely involves a rollback

of the simulation. Pritchett et al. (2001) have demonstrated this method, and named it asynchronous with re-synchronization. They distinguish between complete and partial resynchronization. Complete resynchronization involves the rollback and synchronization of all models involved in the simulation. Partial resynchronization requires only those sub-models to be rolled back and synchronized that experience an effect due to the discrete event that was triggered by the discrete event model(s). Figure 85 depicts the two principles in a notional manner.

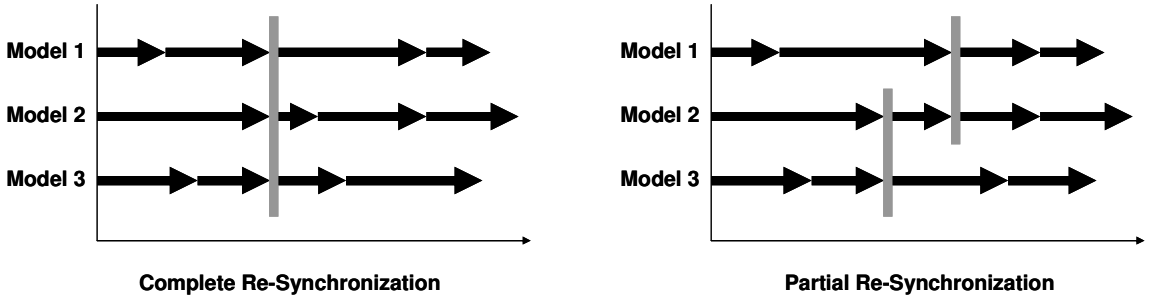

Figure 85. Different ways of discrete even model synchronization with continuous tie models

While the latter approach may be computationally cheaper due to fewer models to be synchronized, issues need to be considered with respect to the necessary data to be exchanged and the sequence in which the models need to be updated. The authors give the following example: If Model 1 requires Model 2 to be updated, Model 2 requires Model 3 to also be updated, and Model 3 requires Model 1 to be updated, then the simulator might be stuck in an infinite loop.

A small side note also comes from the paper by Law and Kelton, namely that of next event time advance. This means that the simulation time step is only updated when a discrete event has occurred. This is of course not applicable here, as the time step for the continuous time models needs to be updated with respect to those model's requirements, and can not consider the discrete event time step.

Naturally, discrete event synchronization may turn out to be difficult if not impossible if the continuous time models are executed with fixed time steps. If the time step can not be changed, e.g. because a sub-model does not allow time step changes "on the fly", then a discrete event will most likely be triggered outside of the fixed time stepping scheme for the continuous time models. In such a case, the synchronization and model updates can naturally still only take place at the fixed time steps. This may lead to the scenario that the discrete event is triggered in between time steps but can only be taken into consideration at the next synchronization and update step. The effects of this delay will vary according to the models and their setups.

It should be noted that strictly speaking, continuous time simulation is to a certain degree also a discrete event simulation type. This is because the time steps are discrete, the necessity of which has been discussed earlier. Therefore, each time step (even in a setup without particular discrete event models) represents a discrete event at which the simulation will stop all or some of the sub-models, synchronize them and exchange their data. The difference is that none of the sub-models contains its own internal clock. Rather, the times when the events re to be triggered is set externally through the scheduler script of the co-simulation, which sets the time steps for the different sub-models and takes care of model synchronization and data exchange.

4.3.6 Selection of RKF23 over other RKF methods

The RKF23 algorithm was introduced to be used for the time step settings in this thesis text. It is the RKF predictor-corrector method of the lowest possible order since it uses only one single predictor and only one single corrector point. This means that it has the least possible amount of function calls, earlier identified and justified as the main performance metric. However, at least one other RKF algorithm has been mentioned previously. The RKF45 method uses 4 predictor and one corrector point, which clearly

requires a much higher computational expense at each point. The question then is why such an algorithm exists and is used (the RKF45 is also widely employed, and implemented in commercial mathematical software packages) if its computational expense is much higher than a similar algorithm of lower order.

To answer this question, consider Figure 86 where both the RKF23 and RKF45 were used to solve a simple sinusoidal differential equation. The same tolerance level was used for both algorithms.

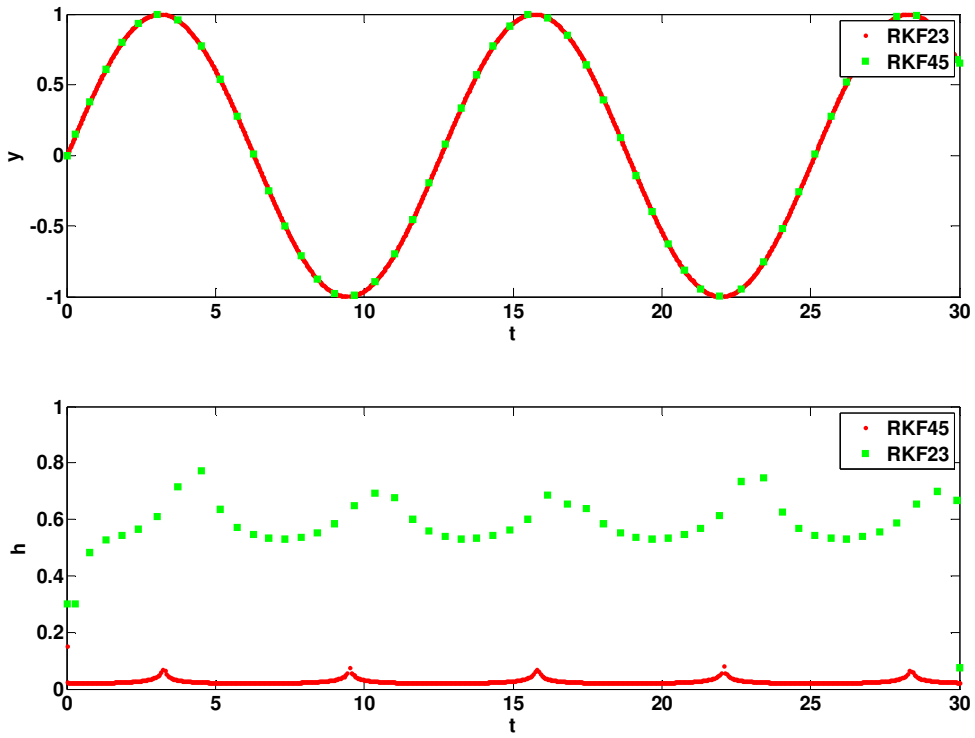

Figure 86. Comparison between RKF23 and RKF45 algorithms for numerical solution of differential equation

It can be readily observed that the RKF45 algorithm allows much higher time steps, and thus requires far fewer actual points to approximate the underlying curve with the same error as the RKF23. In fact, the RKF23 requires such low time steps, and subsequently such large amounts of points, that its overall number of function calls is

much more than the RKF45 algorithm's number of required function calls. However, this behavior does not mean the RKF45 algorithm can be implemented into the co-simulation setup. The reason for this is that when solving an ODE with any of the adaptive time step algorithms, the underlying actual curve is unknown. Hence, in order to approximate this unknown curve, the algorithm not only determines an appropriate time step, but also finds an approximate solution for the next time step. This is done using the different slopes, as described in the explanation of the RKF23 functionality in Chapter 2. This approximated solution is then used as the next point for the solution. In the co-simulation case, the approximate solution is not used, because the underlying function (which is essentially the output result of the sub-model(s)) is known in the sense that it is determined from the simulation and as such serves as the solution to the simulation. Therefore, an approximation is not required. The reason why the time stepping algorithm is employed in the co-simulation is purely to find an appropriate time step, not to determine an approximation solution to the state trajectories. In fact, the approximated solution of the RKF23 is worse than the approximated solution of the RKF45 algorithm due to its lower order. Therefore, the RKF23 must compensate for this by using a smaller time step. The RKF45 algorithm would adapt the time steps as shown before, with very large time steps that would be unacceptable for co-simulation. Therefore, it is not suitable for application in co-simulation.

4.4 Conclusion

This M&S example demonstrated the application of the employed M&S environment for developing and testing of an applied time stepping algorithm to an integrated model of "Black Box" sub-models. It was realized that the underlying problem is similar to that of numerically solving differential equations (DEs). For DEs, the underlying state trajectory is unknown and must be approximated using numerical

methods. In order to control the error, such a method must introduce an adaptive time step. Numerous such algorithms have been developed. This thesis text proposes the use of such an algorithm for the similar case of approximating unknown state trajectories in co-simulation setups. First, a simple toy problem was created in both a monolithic and co-simulated (split and re-integrated) setup. The split model's accuracy compared to the monolithic model, and its robustness against time step changes were evaluated, thus ensuring the fit of the split model for the use of a time stepping algorithm to be developed. The necessity and critical importance of adaptive time stepping for dynamic system M&S in digital computers were explained. Adaptive time stepping helps to reduce the amount of function calls needed to execute a simulation over a given time, thus saving computational expense and finally, money. It helps to reduce this expense without sacrificing result accuracy. It also helps to control the error of the simulation. Since an integrated co-simulation model is a dynamic system itself, time stepping will be critical for such a model as well. The similarities between the numerical solution of ordinary differential equations (as commonly used to describe dynamic systems) and tracking an unknown state path in a co-simulation have been shown. The rationale for using a time stepping algorithm that is proven to work in the numerical integration of ODEs, and applying such an algorithm to co-simulation time stepping followed from this similarity. The chosen time stepping algorithm was a predictor-corrector Runge-Kutta-Fehlberg 2/3 algorithm which represents a good compromise between accuracy and required number of function calls. This algorithm was implemented in the toy problem. The toy problem was expanded by one degree of freedom, to demonstrate that the algorithm is scalable. This expanded model was used to do further investigations with respect to applicability of the adaptive time step vs. a comparable fixed step algorithm. Further applications of adaptive time stepping were pointed out.

The demonstration showed that the developed method is capable of accurately setting a time step in an integrated simulation of dynamic systems. It was shown that this

resulted in reduced amounts of function calls, and a means to determine a proper time step to a system whose dynamics are unknown. Determining a time step for each sub-system, or even for each state variable to be exchanged between sub-models, allows for further reduction of the number of function call to be made since every sub-model is treated in a respective "optimal" way, function call overhead is avoided, and the efficiency of the algorithm further improved.

CHAPTER 5 SUMMARY, LIMITATIONS, AND NEXT STEPS

5.1 Summary

Modeling and simulation have become indispensable tools for system design. Traditional modeling with monolithic models was a first start towards M&S-based design, but proved to have severe limitations that would hinder progress beyond a certain point. Many of these inherent shortcomings of M&S with monolithic models can be overcome by connecting such models together and run them in parallel, with controlled time steps and data exchange. The underlying sub-models need not be known, and indeed their modeling equations and dynamic behaviors are assumed to be unknown for the context of this thesis. Such a setup is referred to as co-simulation. One of the most important parts of such a setup is the scheduler, which controls the entire simulation, data exchange between models, time stepping, execution schedules, etc. The scheduler also contains a database where the state variables of each step are stored.

In computer simulation, real world time can not be represented accurately. Hence, the execution of any simulation must be in discrete time steps. In order for the simulation time to be within reasonable limits, it is desired to adapt the simulation time steps such that the amount of sub-model executions is reduced as much as possible. Yet, it is necessary to keep the model accuracy as high as possible or needed. The best way to control both execution times and model accuracies is by means of variable time stepping.

In order for the time steps to be set appropriately, a method must be found to determine how high the time step can go without serious implications for the accuracy of the results. Until now, the time step is usually set to some arbitrary value, or according to the underlying dynamics of the sub-models. Since these dynamics are assumed to be unknown in the context of this thesis, a different approach must be found towards the

determination of a feasible time step. One such approach is taken from algorithms for the numerical integration of ordinary differential equations (ODEs). This type of equation is often used to model dynamic systems, hence its underlying methodology is assumed to be similar to that of co-simulated dynamic systems. Numerically solving ODEs is necessary in the case where there exists no closed-form solution of the equation. Such algorithms use knowledge of the system states at current past, and future points in time (time is the independent variable) to determine the largest time step that allows the solution to remain within a certain user-specified time step. Running such algorithms for known functions, and recording the chosen time steps over a specific metric gives a mapping that can be used to set the time step for a co-simulation with unknown underlying dynamic system behavior. One such metric is the second derivative, or curvature, of a system state. Intuitively, it makes sense to reduce the time step when the curvature is high, and increase the time step when the curvature is low. However, the second derivative is itself a changing metric for each time step, and will not itself consider changing dynamics sufficiently to be a sole metric for time step setting. Therefore, a more advanced algorithm in the form of the RKF23 is proposed. It provides a means of setting the time step for the integrated model in such a way that an otherwise unknown dynamic system can be handled. The RKF23 does system dynamics "on the fly", taking into consideration the current model behavior and dynamic responses in order to determine a time step that helps to keep the error within certain bounds. The RKF23 application to co-simulation hence allows the time stepping of the integrated simulation despite the presence of "Black Box" sub-models.

The consideration of different dynamics within the different sub-models has proven to lead to reduced number of function calls, and thus a speed up in the simulation execution. A further improvement could be made by taking the curvature into account when applying an error tolerance. It has been mentioned that the RKF algorithm adapts the underlying curve according to a user prescribed error tolerance. The tighter this

tolerance is, the smaller the time steps need to be in order to stay within the prescribed tolerance. However, it has been indicated earlier that when the local curvature is very small (the slope does not change a lot, or: the second derivative of the curve is low), then the tolerance requirement could be relaxed. This would enable to further reduce the computational expense in regions where the local underlying state variable slope does not change much, without allowing too much additional inaccuracies due to the increased time steps. While it would require additional computations to calculate the local second derivative of the state, this is usually a much cheaper calculation than a simulation function execution.

Also, if a model is known to generally behave very smooth and to change this behavior only in special cases, then the application of "stepwise" dynamics might be considered. During times when the model behaves smooth, a fixed time step might be chosen that is cheaper than the RKF application. The RKF would only kick in when the dynamic behavior of the model becomes more erratic, and the time step needs to be adjusted in order for the simulation to stay within error bounds. This would require a cheap metric to determine whether the simulation is still in the smooth range, or whether the algorithm needs to b changed towards the application of RKF. Such a cheap metric could also be the second derivative. It is cheap to calculate, and could give enough indication as to whether the time step needs to be adapted or can stay fixed. Similarly, the RKF calculated deviance between predicted and corrected state can be used as a metric to determine when switching back to a fixed time step would be appropriate.

5.2 Limitations

The algorithm proposed here can not yet be an ultimate algorithm, universally applicable for all possible dynamic system co-simulation scenarios. It merely presents and proposes an approach towards the underlying problem "Co-Simulation of 'Black Box' dynamic models", an issue which in this way has not been tapped into by common

literature as of yet and as of the author's knowledge. The proposed algorithm was developed and verified using a simple, well-understood model as the underlying test subject. This simple model did not show many of the underlying problems that co-simulation can be plagued by, as described in a previous chapter. It was simple, well-behaved, and robust. It must be kept in mind that treating an ODE is not the same as treating a simulation. This can be easily seen by the following consideration: With the form of an ODE according to Eq. 1, any given time step and state variable magnitude (again neglecting external inputs) will always result in the same result for the slope. This is the case because the underlying relation between slope and independent variables is a fixed equation. However, a simulation does not behave in such a manner. It is easy to understand that the slope and curvature of a simulation can not be evaluated deterministically like an equation. When evaluating the slope of a simulation at the current point, the result will depend on previous simulation states (this is valid for any simulation, even if it is a monolithic model with only one state variable). This is especially valid with respect to variable time steps. The current system behavior will strongly depend on how far back the previous time step has been, and how much "dynamics" play into the model during the previous and current time step. Furthermore, when using a future point for the calculation of a second derivative, this point will also depend on the dynamics and current time step settings. If the proposed algorithm is to be used in other co-simulation scenarios where the underlying sub-models and/or the integrated model pose problems, further investigations and improvements to the integration algorithm must be made. Some of these suggested improvements are listed below.

5.3 Next Steps

The algorithm in its current form is not generally applicable to general co-simulation setups. This is due to a variety of reasons. In general, since a simulation generally does not behave like an equation (i.e., deterministic), the results at each time step depend on the previous time steps and system states. The next steps are intended to suggest measures to be taken in order to improve the approach, to make it as generally applicable as possible, and to try to reduce time requirements and to increase accuracy. Several steps can be considered:

- As mentioned in the discussion about the results in Chapter 4, the scope of this text allowed only for a limited amount of testing and evaluation. Clearly, there could be more examinations done. Both masses and bar lengths could be varied, in a wider range than shown here. Also, the initial conditions (starting angles and accelerations) could be varied. The algorithm could be implemented in other dynamic models, e.g. as shown in Figure 73 and Figure 74. The algorithm scalability would also have to be investigated. Initially used for only the minimum number of two sub-models for a co-simulation, this was extended to three such sub-models. However, coupled model simulations may require the inclusion and treatment of multiple sub-models, for example in planetary research with multiple planets and objects whose trajectories need to be established. This is clearly one of the major future tasks to be performed. The results presented in Chapter 4 may hold only under certain circumstances not covered in more depth in this text. However, the general implementation approach of an adaptive time step algorithm that considers the individual dynamics of the sub-models is deemed to be applicable even for extended co-simulation setups with multiple sub-models, as long as the data synchronization and trajectory data exchange are taken care of rigorously.
- Inherent to the RKF23 algorithm, the prescribed error tolerance is an *absolute* error. However, depending on the absolute magnitude of a state variable, certain

absolute error bounds might not be feasibly achievable without invoking extreme, and thus prohibitive, computational expenses. Therefore, for the RKF23 test runs, as well as for any actually applied time stepping algorithm, a *relative* error should be applied. Some metric must be determined to determine how to set this error, and invoke it into the algorithm.

- With respect to tolerance, one could introduce an adaptive tolerance. Currently, the RKF tolerance is prescribed at the start of the simulation and remains constant throughout. It could be seen particularly from the results presented for the triple pendulum, that occasionally there are deviations from the "right" solution, the results for the fixed step setup. In such cases, the tolerance level of the RKF implementation is likely to be insufficient to keep the results accurately enough. An adjusted tolerance level, e.g. as a function of the current frequency or second state derivative, might help to stay mode accurate in regions where higher tolerance is required. It must be taken into account though, that this will likely increase the overall number of function calls. While the minimum time step is the step applicable to the fixed step split model in order to have comparable accuracies throughout the simulation run, and hence increased function count on the side of the fixed-step split model as well, careful consideration must be given to this setup as it may turn out that under certain circumstances the fixed step setup will prove to be advantageous with respect to performance where it had previously (with constant tolerance) been at a disadvantage.
- It has been mentioned above, that when very high frequencies exist within the simulation (see Figure 75) there will be a frequency shift between the fixed step and adaptive step models. This shift is not constant, and its cause is not yet understood. Such behavior warrants further investigations in the causes and the potential remedies of this effect.

- The second derivative, being an intuitive metric as a time step setting criterion, may not be a sufficient metric. As has been mentioned, the time step resulting from RKF23 runs with known functions did not always result in equal time steps for equal local state variable curvatures. This is due to the fact that the curve requires different time steps depending on its current degree of fluctuation. A pure second derivative however, does not take this current behavior into account. Therefore, a more sophisticated approach will likely be necessary. Local degree of fluctuation may be captured by higher-order derivatives. It makes intuitive sense that for example, a quick change in curvature (which would be equal to the local third derivative of the curve) might have an impact on the necessity of smaller time steps. This will lead to the issue of how to calculate such a derivative. Currently, the second derivative is calculated with one step in the past, the current step, and one step in the future. It therefore takes the future development into account. A third derivative could be calculated e.g. by using 2 past points, the current point, and one future point. It could also however, be calculated using one past point, the current point, and two points in the future. This would take the current behavior into account even better. Such an approach would come very close to an approach similar to the RKF23 algorithm: Take steps in the future together with the current step (and some past steps as well, an approach taken by various Adams numerical integration methods, which retain the past information for a more accurate slope and step estimation), and based on the results determine a "best" time step. This might be achieved by using less future time steps than RKF45, and lead to a simpler algorithm. It has been discussed that such an approach, like any approach using future points, will require increased amounts of function evaluations, a fact that runs detrimental to the requirement of reduced execution time.

- The previous point leads to another way of looking at the problem of adequate function behavior evaluation. Since steps in the past and the current step are already available (stored in the co-simulation scheduler database), it may make sense to use Splines, extrapolation, or forecasting to calculate a range where the next point is expected to be. A method that applies this approach is the Richardson extrapolation (Richardson, 1911), which has specifically been developed for implementation in numerical integration of DEs. While developed in 1911, this method has proven its worth over time, and has been both developed further and used in multiple applications (see, for example, Lether (1965), Tseng and Lee (2008), and González-Parra et al. (2010)). This could be with an error bound. If the actual next point is within the given error bound of the expectation, then the point is accepted, and the time step unchanged. If however, the next point lies outside the expected error range, then it means that the time step was too high, and must be corrected before a new point can be set. This approach could be executed multiple times, in order to achieve a given error bound, but as usual at the expense of an increased number of function calls. An extrapolation algorithm would also possibly bypass one of the main problems that occur when using numerical integration algorithms on "Black Box" models: The algorithm will always require the slope of the state variables which, under the "Black Box" assumption, is not directly available. Any approximation of the slope will introduce errors. An extrapolation algorithm may not require the slope any more, and work solely through the state variable values directly. This would also help to reduce the computational expense of the simulation. Figure 87 depicts the approach. Any such approach would almost necessarily fall into the category of predictor-corrector method. When discussing the RKF23 and similar time stepping methods before, it was already stated that in order to control error, the time step can not be constant but must be adaptive. All of the established and

proven methods for numerical integration look into the future to some extend in order to determine the behavior in the immediate future, and to determine the "best" time step for the next point. The extrapolating / forecasting approach is just a different way of looking at the problem. Instead of calculating several points in the future and inferring the system behavior before setting a new time step for the next point, extrapolating / forecasting simple calculates the next step and then checks whether it falls into the expected range before adapting the time step as necessary. The time step could possibly be estimated through a simple interpolation and evaluation of the crossing point of the extrapolation with the prescribed error bound. This has the advantage that if the time step needs not be corrected, the calculated point can then be used as the next point right away, without any additional computational expense. A new time step, and hence a new point, is only calculated if the actual future point fall out of the specified error bound. However, there is no criterion regarding the *increase* of the time step for slow-changing curvatures. This would require an adaption of the algorithm. Further, it must be kept in mind that extrapolation has been adapted for non-stiff systems, but is a bit more problematic for stiff systems. Hairer and Wanner (1996) discuss this problem, and offer solutions on how to use extrapolation for stiff systems as well. As will all literature however, they assume the system equations to be known. Pham and Oudin-Dardun (2009) describe the use of extrapolation and time stepping in a co-simulation setup that is distributed over several parallel computers, in order to share the computational loads and increase the speed of execution. Their approach goes into the more complicated issues of solving ODEs and DAEs in nonlinear systems. Their setup consists of coupled sub-systems of equations, and they investigate performing asynchronous communications to send the data needed for extrapolation. The paper gives details of implementation as well as numerical and efficiency performances. As above, the underlying

equations are known. Even though these equations are not given in the context of co-simulation as discussed in this text, this seems to be an approach worth investigating.

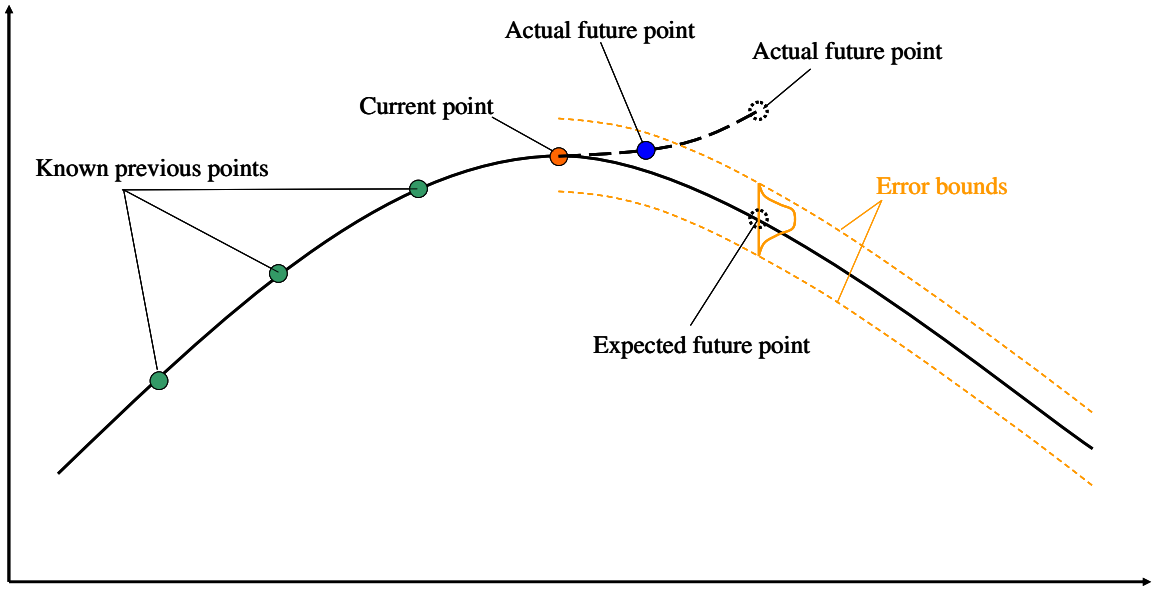

Figure 87. Proposed extrapolation algorithm for time stepping

- One input parameter that is required for an RKF algorithm is the maximum time step allowable. It was discussed previously that the time step is very critical to not only the accuracy of the results, but also the stability of the simulation against changes in initial conditions. Since the result of every time step run is the initial condition for the next time step run, such errors can quickly grow beyond control. If a system has very "slow" dynamics, the time step can become larger subsequently. However, it must be made sure that the time step does not grow so large that it leads to instability of the simulation. Therefore, the time step size must be limited to prevent this from happening. In order to have an overall idea of how large the time step should be set as a maximum, one needs some information regarding the dynamics of the underlying sub-models. One way of doing this is to run the models in separation and to feed the model's inputs with signals that are similar to the expected signal inputs under co-simulation runs. The outputs of the

sub-models can then be recorded. The recorded data are analyzed using a Fast Fourier Transform (FFT). This will show the dominant frequencies of the system, based upon which a rough estimate of the maximum time step can be made. Figure 88 shows an FFT output of the upper pendulum bar with a "slow" dynamic setting.

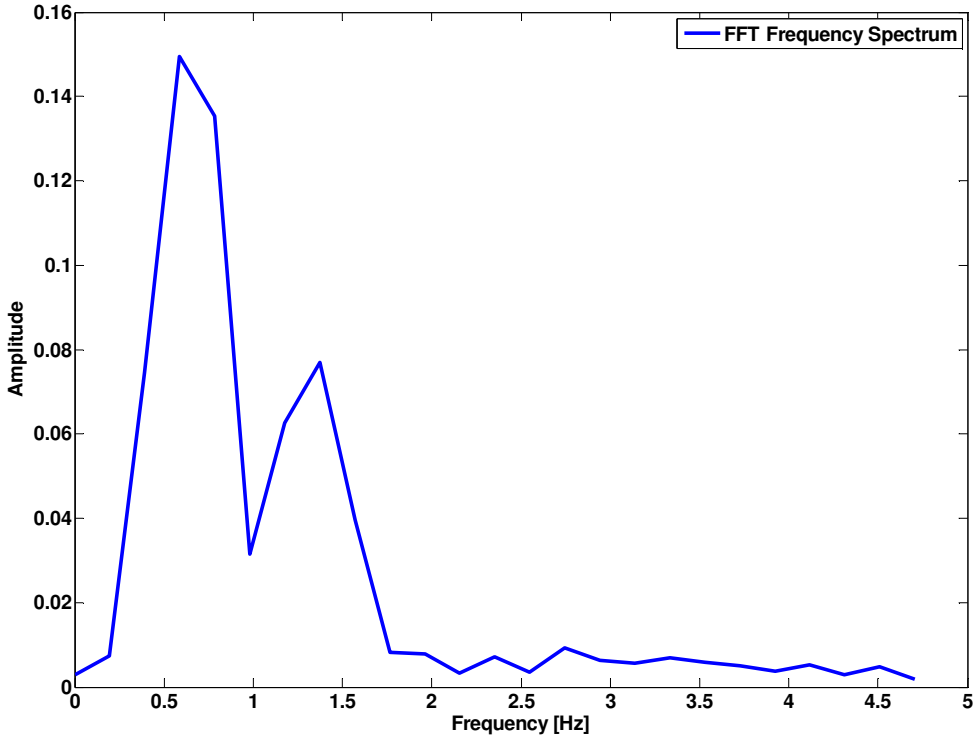

Figure 88. Sample FFT result for upper pendulum bar

The highest peak of the dominant frequency can be observed at a frequency of 0.7 Hz. Therefore, the half wavelength is 1.4 seconds, and the wavelength is 2.8 seconds. This agrees well with Figure 46, where a low at 1.2 seconds and a high at 2.7 seconds can be observed. From this, a maximum time step can be derived. This is currently under investigation, as the Nyquist criterion would result in too high an allowable maximum time step for this simulation setup. However, the

FFT approach promises to shed enough insight into the dynamic behavior of the underlying sub-systems that it allows for determination of a maximum time step.

- In order to determine whether a variable time step falsifies the outputs from any sub-model, it may be a good approach to feed sub-models with known input commands and record their behavior and reactions to those commands. Using these results, one can then determine how a sub-model, and the whole co-simulation setup, behaves when similar inputs are used for the sub-models, and the time step is varied according to some algorithm. This would help to determine the error introduced into the co-simulation due to the adaptive time step. Figure 89 shows a notional graph to clarify the approach.

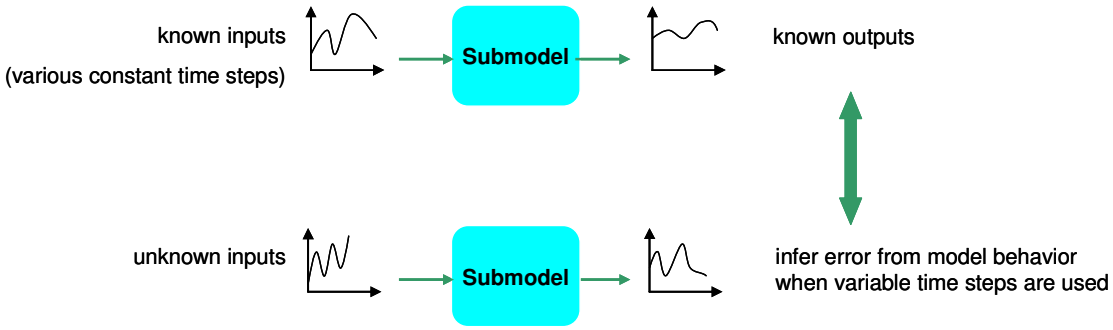

Figure 89. Method to estimate error due to adaptive time step

- From the discussion of the general approach, it is easy to understand that at each time step, there will be some error introduced into the system (except in the case where the second derivative is equal to zero; in this case the error will remain constant). This error will accumulate over time. This beckons several issues to be investigated. For example, as seen in Figure 11 and in the discussion about Euler's method, the error will be positive if the underlying state variable curve is concave and negative if the curve is convex (see the discussion on Euler's method for definitions of convexity and concavity). However, in a realistic physical dynamic system, no curve can be convex or concave forever. This means that as soon as the curvature changes its sign, the current accumulated error will start to

decrease again. So the question is then, will there be a natural limit to the error? No matter the answer to this question, the long-term behavior of the system must be evaluated. Even if the curvature changes at some point in time, until that time the error through introduced noise accumulates, and the simulation may have to be restarted at a certain time with a known state. But then the question is what criterion to use to determine when the simulation should be stopped and restarted? More investigations into the long term behavior of co-simulated systems would be necessary to evaluate such a criterion, especially under the consideration of variable time steps and error control. Calvo et al. (2008) aim to show with their paper the behavior of first integrals of some differential systems integrated by several Runge-Kutta methods over long term intervals. It studies the growth of the error in the first integrals of ODEs that possess periodic orbits for general RK methods, and gives results and explanations for a number of numerical experiments for geometric symplectic and pseudo-symplectic numerical integrators. As always, the underlying equations are known, and the method could therefore only serve as a basis for a similar approach. Utumi et al. conclude that based on the fact that the error accumulates over time, using only local information to adapt the current time step size may not be the optimal solution, but that rather a global error criterion may be needed. They represent a solution based on the basic works of Morrison (1962), Greenspan et al. (1965), Gear (1971), Lindberg (1977), and Butcher (1986), and by using methods and concepts from optimal control. Their claim is that they can determine an “optimal” time step for a given error and with respect to a minimum execution time requirement. In this paper, just like in most other papers presented in this thesis, the underlying equations are assumed to be known. Yet, this idea of the local error criterion being sub-optimal for time step determination may have great potential to be included into further such research for the application in co-simulation. Rios Neto

and Rama Rao (1990) discuss the inclusion of an implementation of stochastic approaches towards global error estimation and the determination of the integration time step for known equations, specifically of the Adams-Bashforth-Moulton type of multistep methods. Johnson (1988) introduces a priori error estimates for a class of implicit one-step methods for stiff ordinary differential equations. This specific approach is developed with respect to discontinuous Galerkin methods with piecewise degrees zero and one (Galerkin methods are a class of methods for converting a continuous operator problem, such as a differential equation, to a discrete problem). While they propose a new time step control algorithm, the underlying Galerkin method makes this approach less suitable for the current problem under investigation.

- The IRIS system, which was the initial motivation for this research, was planned to simulate the interconnected systems on the new Navy battleship, DD(X). As such, this system is a military application, and any simulation must take into account the possibility of external shocks to the system, most likely through enemy fire impact. However, this is also interesting for civilian applications, since self-repairing systems will also need to be tested for sudden system disruptions, such as damages through explosions etc. Both types of systems will also need to be able to reliably handle external inputs by users, which could also have a sudden and disrupting effect. In any case, this represents an external shock input to the system, and the simulation will need to be able to cope with such an incident. After all, this is what one of the motivations for simulation was from the beginning: Being able to simulate such shocks without having to destroy an actual physical test system. External shock inputs will change the state variables that are exchanged between the systems in an almost step-function-like style (a true step function is of course not possible, since every time step will be finite and larger than zero, hence the slope can never reach infinity). The simulation must be able

to determine that such an incident happened, and must also be able to distinguish whether it could be a normal working state of the system, or whether this event must be an external shock. In case of an external shock, since the system disruptions will be severe, the time step will need to be very small. However, the event is now known beforehand, and the time step correction might come too late. In such a case, one solution would be to "roll back time" to the previous time step, and rerun the simulation with a very small time step. Co-simulation allows for this because all the previous system states are stored in the scheduler database, and can be used again to restart the simulation at any given point without and impact on the results. The simulation can then be restarted at an instance before the (now known) impact, and the time step can be set to be small enough for the simulation to cope with the shock. The principle of rollback is applied e.g. in Yoo and Choi (2000), but their principle is somewhat different because it deals with hardware-software co-simulation (a different definition of co-simulation), and requires the synchronization between the hardware and the software. It is also more of a discrete simulation because discrete time events for data exchange are defined. Nevertheless, rollback is a practical and applied method for simulation that greatly enhances its applicability for such problems.

- In general, the co-simulation time stepping will need to be tested under certain extreme conditions in order to find out the limitations of adaptive time steps. These limits will likely be under very small and/or very high curvatures of the underlying state variables. In the case of very low curvatures, the time step can in theory be very high (if all state variables under consideration have low curvature). But this has the inherent danger that small changes in the system might be overseen, and their effects be neglected from the simulation. This is not acceptable, and hence the time step must have an upper limit whose magnitude remains to be determined. A similar problem exists with the lower time step limit.

How low can the time step become before numerical and truncation errors become too large and start to falsify the simulation results?

- With time stepping in co-simulation only discussed in the literature with the mathematical equations of the sub-models and/or the overall integrated simulation given, it would be helpful to find out about the sub-systems behaviors in more detail, in order to be able to treat the integrated simulation with even more mathematical rigor. As discussed, system identification tools are available, but are limited to linear models. Hence, the linearity must be assumed or given in order to be able to apply these techniques. State observers are a concept from control theory. They extend the more specific technique of system identification, which itself is limited to linear models. State observers are mathematical systems that model a real system so as to be able to provide an estimate of the real system's internal states, given measurements of the input and output of the real system. Knowing the real system's internal states is necessary to solve many control theory problems; for example, stabilizing a system using state feedback. In most practical cases, the physical state of the system cannot be determined by direct observation. Instead, indirect effects of the internal state are observed by way of the system outputs. If a system is observable, it is possible to fully reconstruct the system state from its output measurements using the state observer. Observers and their design are described e.g. in Meurer et al. (2005) and Schröder et al. (2010). Lin et al. (2008) present the necessary and sufficient conditions under which a discrete time autonomous system with outputs is locally diffeomorphic to an output-scaled linear observable system or an output-scaled nonlinear system in the observer form. They study the non linear observer design problem as a consequence of such characterizations, and use a time scaling approach combined with the exact linearization technique, for a broader class of discrete-time nonlinear systems. Karafyllis and Kravaris (2008) develop a sampled data

nonlinear observer using a continuous time design coupled with an inter-sample output predictor, leading to a hybrid system. They show that under certain conditions, the robustness properties of the continuous time design are inherited by the sampled-data design, as long as the sampling period is not too large. The approach is applied to linear systems and to triangular globally Lipschitz systems. Califano et al. (2010) aims to generalize results from recent papers on observer design to a wider class of discrete time systems affected also by the control and by considering a more general output scaling structure. They present a constructive proof, thus allowing a straightforward computation of the desired observer. In general, observers could be used to shed more light into the general behavior and the "internals" of the sub-models used. Such an approach would help to make "Grey Boxes" out of the currently investigated "Black Box" approach. With more knowledge about the internals of the sub-models, more rigorous mathematical treatment of the time stepping problem could be initiated, coming closer to the earlier introduced methods that allow time step setting using the known equations that describe the sub-models completely. This could lead to an extension of the current method. For example, if the state derivatives could be given by the sub-models as a result of turning the "Black Boxes" into "Grey Boxes", then the time step problem in essence becomes a problem of solving a system of systems of differential equations (Case 3 in Table 1), which in turn could be tackled by applying algorithms for the time stepped solution of systems of differential equations, for which strict mathematical routines exist.

- The proof of concept presented in this text was done with simple dynamic models to try a first approach. Naturally, one very important step with any algorithm developed in the context of this research will be to try the developed method for as many different applications as possible to determine its applicability to different scenarios and situations.

- Lastly, the methods described here were based on numerical integration schemes for ODEs. While this is somewhat intuitive, given the dynamic nature of the underlying systems, there may be other engineering disciplines where to look for possible solutions. For example, PDEs usually have multiple variables and dependencies (similar to co-simulation), and special multi-stepping algorithms for the solution of PDEs have been developed, e.g. in Sigal (2005). These algorithms may be a better fit for the problem at hand. Such methods are developed for example in Hairer and Wanner (1996), but have the underlying assumption that the system equations are known. As stated multiple times before, in the context of this thesis, the assumption is that sub-model equations, or a possible equation for the over integrated system, are not known. Hence, any such methodology can only serve as a basis for the development of an algorithm that is applicable for such a problem. Also, an investigation into model-based control algorithms may turn out to be fruitful due to the similarity of the problem. Generally, a more inter-disciplinary approach and "out of the box" thinking approach will be necessary to at least determine what other methods are being used in different disciplines.

5.4 Conclusion

The long discussion in the previous section about future work shows that there are a lot of possible directions to look into for the practical application of adaptive time steps. The particular difficulty is that the sub-systems equations and dynamic behaviors are unknown. This case has not been treated in the literature so far. This research will look into different directions and try to develop a method that is universally adaptable for co-simulation of dynamic third-party proprietary sub-models, using adaptive time stepping to increase run time and accuracies. The extensive literature research has shown that there is no commonly known and applied method for such a problem. Any method developed,

proposed, or employed in the literature has its underlying system equations known. In case of co-simulation of integrated sub-models, some papers employ time stepping only for the solution of the sub-models, but not for the overall co-simulation system setup. All these setups are not representative for the problem at hand. However, many of the proposed algorithms and approaches e.g. for error estimation and propagation, and time step adjustments, could serve as a basis for such algorithms that will be suitable for co-simulation setups in the sense of this thesis. Since the importance and complexities of co-simulations will likely increase further, reducing the execution expenses and increasing the accuracies are highly desirable goals for further research and the development of an easily adaptable solution.

It must be noted that, while one reason for using an adaptive time step is a possible reduction in run time for the execution of the co-simulation model, the main point of this thesis is not to achieve real time co-simulation. The implications of real time simulation are much more complex than the issues covered in this thesis, since it requires clocking of the simulation, and more control over the sub-models and their execution speed. Since the sub-models are assumed to be "Black Boxes" with no access to their "interior", such control is not given, and real time execution speed can not be achieved. Hence, the method developed in this text is solely for the purpose of preliminary design, not operational applications. If co-simulation system execution speed is the primary interest, then other methods exists, such as sub-model representation through surrogate models (which themselves would introduce errors), or other methods as described in e.g. Cuadrado et al. (1999) where the authors put the (known) system equations in different forms for faster and more accurate treatment using a multi-index variable time step method for the integration of the equations of motion of constrained multi-body systems in descriptor form. Interestingly, this paper also uses a double pendulum for evaluation of the method.

APPENDIX A RKF23 META CODE

```
set T = start time
ΔT = some very small time step for high accuracy
set τ as error tolerance
set x as stating point (initial condition)
set h_max as maximum time step allowable
while T<= end time
     determine slope k1 at current point
     determine slope k2 at the point T + ΔT/2
     determine slope k3 at the point T + ΔT
     calculate predictor point x2 with slope k2 as x2=x +
ΔT * k2
     calculate corrector point x3 with slopes k1, k2, k3 as
x3=x + ΔT * (1/6*k1 + 4/6*k2 + 1/6*k3)
     calculate prediction error as γ1 = x3 – x2
     if γ1 < τ
          let x = x3
          let T = T + ΔT
     end if
     let ΔT = min(h_max, 0.8*ΔT *(τ / γ1)^0.25)
end while
```

APPENDIX B RKF23 CODE FOR TESTING PURPOSE

```
% Runge-Kutta-Fehlberg 2/3 (RKF23) variable time step
algorithm

% This sample algorithm uses a known and given function and
its known
% and given derivative (= slope) to verify the RKF23
algorithm

% adapted from the original posted at:
%
http://cns.bu.edu/~tanc/pub/matlab_octave_compliance/ode_v1
.11/ode23.m

clc; clear all; close all; format long; % prepare Matlab
environment

tol = 1.e-3 % desired tolerance level
ff=2 % curve amplitude parameter
t=0; % starting simulation time
tspan=[t 100]; % simulation time span
x0=(sin(sin(t/23)*t))/(sin(t)+ff);    %    exact    solution
equation
x_bar=[(cos(t*sin(t/23))*(sin(t/23)                        +
(t*cos(t/23))/23))/(sin(t)          +          ff)          -
```

```
(sin(t*sin(t/23))*cos(t))/(sin(t)   +   ff)^2;];   %   exact
solution slope equation

rhs_counter = 0; % counter for number of function calls

x_exact=[x0]; % vector with exact state values

% for the following power factor, see p.91 in:
% Ascher, U. M.; Petzold, L. R. (1998)
% "Computer Methods for Ordinary Differential Equations and
Differential-Algebraic Equations"
% SIAM, Philadelphia, ISBN 0-89871-412-5
pow = 1/4;

% RKF23 coefficients:
a(1,1)=0;
a(2,1)=1/2;
a(3,1)=-1; a(3,2)=2;
% 2nd order b-coefficients
b2(1)=0; b2(2)=1;
% 5th order b-coefficients
b3(1)=1/6; b3(2)=2/3; b3(3)=1/6;
for i=1:3
c(i)=sum(a(i,:));
end

% Algorithm initialization
```

```
t0 = tspan(1);
tfinal = tspan(2);
t = t0;
hmax = (tfinal - t)/2.5; % determine maximum time step
hmin = (tfinal - t)/1e12; % determine minimum time step
h = (tfinal - t)/200; % initial guess at a step size
x = x0(:)             % this always creates a column vector,
x
tout = t;              % first output time
xout = x.'            % first output solution
h_store=[h]; % store time steps

% The main loop
while (t < tfinal) & (h >= hmin)
  if t + h > tfinal, h = tfinal - t; end

  t1=t+c(2)*h; % first RKF23 future time point
  t2=t+c(3)*h; % second RKF23 future time point
  k1=(cos(t*sin(t/23))*(sin(t/23)                          +
(t*cos(t/23))/23))/(sin(t)          +          ff)          -
(sin(t*sin(t/23))*cos(t))/(sin(t)   +   ff)^2;   %   slope   at
current point
  k2=(cos(t1*sin(t1/23))*(sin(t1/23)                       +
(t1*cos(t1/23))/23))/(sin(t1)         +         ff)         -
(sin(t1*sin(t1/23))*cos(t1))/(sin(t1)  +  ff)^2;  %  slope  at
first future point
```

```
  k3=(cos(t2*sin(t2/23))*(sin(t2/23)                          +
(t2*cos(t2/23))/23))/(sin(t2)        +        ff)        -
(sin(t2*sin(t2/23))*cos(t2))/(sin(t2) + ff)^2; % slope at
second future point

  rhs_counter = rhs_counter + 3; % update function call
counter

  x2=x + h*b2(2)*k2; % compute the 2nd order estimate

  x3=x + h*(b3(1)*k1 + b3(2)*k2 + b3(3)*k3); % compute the
3rd order estimate

  gamma1 = x3 - x2; % estimate the local truncation error

  % Estimate the error and the acceptable error
  delta = norm(gamma1,'inf');
  tau = tol*max(norm(x,'inf'),1.0);

  if delta <= tau % Update the solution only if the error
is acceptable
     t = t + h; % update current simulation time point
     x = x3;     % <-- using the higher order estimate is
called 'local extrapolation'
     tout = [tout; t]; % store simulation time point
     xout = [xout; x.']; % store RKF23 state variable value
approximation
```

```
    x_exact=[x_exact   (sin(sin(t/23)*t))/(sin(t)+ff)];   %
store exact solution at current time step
    x_bar=[x_bar  k1];  %  store  exact  slope  at  current
simulation time point
    h_store=[h_store h]; % store simulation time step

  end

  % Update the step size
  if delta == 0.0 % keep a minimum error
   delta = 1e-16;
  end

  h  =  min(hmax,  0.8*h*(tau/delta)^pow);  %  calculate  new
time step size

end;

if (t < tfinal) % error if time step got too small
  disp('Step size grew too small.')
  t, h, x
end

rhs_counter % display the number of function calls

% plot the outputs
```

```
figure(1)

subplot(3,1,1)
plot(tout,                                          xout,'--
rs','LineWidth',1,'MarkerEdgeColor','k','MarkerFaceColor','
g','MarkerSize',2)
hold on
plot(tout, (x_exact)','g-')
xlabel('\fontsize{16}\bfT [s]')
ylabel('\fontsize{16}\bfState Exact vs. RKF23')
legend('\fontsize{16}\bfx(RKF23)','\fontsize{16}\bfx(Exact
solution)')
line([0 max(tspan)],[0 0],'Color','r')
plot(tout,                                          xout,'--
rs','LineWidth',1,'MarkerEdgeColor','k','MarkerFaceColor','
g','MarkerSize',2)
plot(tout, (x_exact)','g-')
set(gca,'FontSize',16,'FontWeight','Bold')

subplot(3,1,2)
x_error=x_exact'-xout;
x_relerror=x_error./x_exact.';
plot(tout,x_relerror)
xlabel('\fontsize{16}\bfT [s]')
ylabel('\fontsize{16}\bfRel. Error')
hold on
```

```
line([0 max(tspan)],[0 0],'Color','r')
plot(tout,x_relerror)
set(gca,'FontSize',16,'FontWeight','Bold')

subplot(3,1,3)
plot(tout, h_store)
xlabel('\fontsize{16}\bfT [s]')
ylabel('\fontsize{16}\bfTime Step')
set(gca,'FontSize',16,'FontWeight','Bold')
```

APPENDIX C META CODE FOR INDIVIDUAL TIME STEPPING APPROACH

Meta code for individual time stepping approach

```
Set final simulation time T_Final
start at simulation time T_u = T_l = 0 (T_u / T_l: upper/lower
bar simulation time)
run both bars with RKF23 time step
while T_u <= T_Final and T_l <= T_Final
     if T_u < T_l
          while T_u < T_l
               run upper bar with RKF23
               update T_u
          end while
     end if
     if T_l < T_u
          while T_l < T_u
               run lower bar with RKF23
               update T_l
          end while
     end if
end while
```

APPENDIX D EQUATIONS OF MOTION FOR THE TRIPLE PENDULUM

```
a: alpha, angle of bar 1
b: beta, angle of bar 2
g: gamma, angle of bar 3
a': first derivative of alpha
b': first derivative of beta
g': first derivative of gamma
a'': second derivative of alpha
b'': second derivative of beta
g'': second derivative of gamma
l1: length of bar 1
m1: mass of bar 1
l2: length of bar 2
m2: mass of bar 2
l3: length of bar 3
m3: mass of bar 3
grav: gravitational constant

a'':
Numerator:
(-m2*l2*cos(a-b)*m3*l3*g'^2*sin(b-g)
+2*m2*l2*cos(a-b)*m3*l1*a'^2*sin(a-b)
-m3^2*l2*cos(a-b)*cos(b-g)*l1*b'^2*sin(b-g)
-m3^2*l2*cos(a-b)*cos(b-g)*l1*a'^2*sin(a-g)
+m3^2*l1*cos(a-g)*cos(b-g)*l3*g'^2*sin(b-g)
-m3*l1^2*cos(a-g)*cos(b-g)*m2*a'^2*sin(a-b)
-m2*l2*b'^2*sin(a-b)*m3*(cos(b-g))^2*l1
+m3*l1*cos(a-g)*b'^2*sin(b-g)*m2*l2
+m3*l1*cos(a-g)*a'^2*sin(a-g)*m2*l2+2*m3*l2^2*b'^2*sin(a-b)*m2
+m3*l2^2*g'^2*sin(a-g)*m2+m3^2*l2^2*b'^2*sin(a-b)
+m2^2*l2^2*b'^2*sin(a-b)+m3^2*l2^2*g'^2*sin(a-g)
-m3^2*l2*b'^2*sin(a-b)*(cos(b-g))^2*l1
+m2^2*l2*cos(a-b)*l1*a'^2*sin(a-b)
-m3^2*l2*cos(a-b)*l3*g'^2*sin(b-g)
+m3^2*l2*cos(a-b)*l1*a'^2*sin(a-b)
-m3^2*l1^2*cos(a-g)*cos(b-g)*a'^2*sin(a-b)
-m3^2*l2*g'^2*sin(a-g)*(cos(b-g))^2*l1
+m3^2*l1*cos(a-g)*b'^2*sin(b-g)*l2
+m3^2*l1*cos(a-g)*a'^2*sin(a-g)*l2
-m2*l2*cos(a-b)*m3*cos(b-g)*l1*b'^2*sin(b-g)
-m2*l2*cos(a-b)*m3*cos(b-g)*l1*a'^2*sin(a-g)
+m2*l2*cos(a-b)*m3*cos(b-g)*grav*sin(g)
+m3*l1*cos(a-g)*cos(b-g)*m2*grav*sin(b)+m2^2*grav*sin(a)*l2
+m3^2*grav*sin(a)*l2-2*m2*l2*cos(a-b)*m3*grav*sin(b)
+m3^2*l2*cos(a-b)*cos(b-g)*grav*sin(g)
```

```
+m3^2*l1*cos(a-g)*cos(b-g)*grav*sin(b)
-m1*grav*sin(a)*m3*(cos(b-g))^2*l1
-m2*grav*sin(a)*m3*(cos(b-g))^2*l1
-m3*cos(a-g)*grav*sin(g)*m2*l2-m2^2*l2*cos(a-b)*grav*sin(b)
-m3^2*l2*cos(a-b)*grav*sin(b)+m1*grav*sin(a)*m2*l2
+m1*grav*sin(a)*m3*l2+2*m2*grav*sin(a)*m3*l2
-m3^2*grav*sin(a)*(cos(b-g))^2*l1-m3^2*cos(a-g)*grav*sin(g)*l2)

Demoninator:
(l1*(m3*(cos(a-g))^2*m2*l2+m3^2*(cos(a-g))^2*l2-m1*m2*l2
-m1*m3*l2+m1*m3*(cos(b-g))^2*l1-m2^2*l2-2*m2*m3*l2
+m2*m3*(cos(b-g))^2*l1-m3^2*l2+m3^2*(cos(b-g))^2*l1
-m2*l2*cos(a-b)*m3*cos(b-g)*cos(a-g)+m2^2*l2*(cos(a-b))^2
+2*m2*l2*(cos(a-b))^2*m3-m3^2*l2*cos(a-b)*cos(b-g)*cos(a-g)
+m3^2*l2*(cos(a-b))^2-m3*l1*cos(a-g)*cos(b-g)*m2*cos(a-b)
-m3^2*l1*cos(a-g)*cos(b-g)*cos(a-b)))

b'':
Numerator:
-(-sin(b-g)*g'^2*l3*m3^2
+sin(a-b)*a'^2*l1*m3^2+sin(a-b)*a'^2*l1*m2^2
-cos(a-g)*sin(a-b)*cos(b-g)*b'^2*l2*m2*m3
+cos(a-b)*cos(a-g)*sin(b-g)*b'^2*l1*m2*m3
+cos(a-b)*cos(a-g)*sin(a-g)*a'^2*l1*m2*m3
-sin(b-g)*g'^2*l3*m2*m3-sin(b-g)*g'^2*l3*m1*m3
+2*sin(a-b)*a'^2*l1*m2*m3+sin(a-b)*a'^2*l1*m1*m3
+sin(a-b)*a'^2*l1*m1*m2-cos(b-g)*sin(b-g)*b'^2*l1*m3^2
-sin(a-g)*cos(b-g)*a'^2*l1*m3^2+cos(a-b)*sin(a-g)*g'^2*l2*m3^2
+cos(a-b)*sin(a-b)*b'^2*l2*m3^2+cos(a-b)*sin(a-b)*b'^2*l2*m2^2
+(cos(a-g))^2*sin(b-g)*g'^2*l3*m3^2
-(cos(a-g))^2*sin(a-b)*a'^2*l1*m3^2
-cos(b-g)*sin(b-g)*b'^2*l1*m2*m3
-cos(b-g)*sin(b-g)*b'^2*l1*m1*m3
-sin(a-g)*cos(b-g)*a'^2*l1*m2*m3
-sin(a-g)*cos(b-g)*a'^2*l1*m1*m3
+cos(a-b)*sin(a-g)*g'^2*l2*m2*m3
+2*cos(a-b)*sin(a-b)*b'^2*l2*m2*m3
-cos(a-g)*sin(a-g)*cos(b-g)*g'^2*l2*m3^2
-cos(a-g)*sin(a-b)*cos(b-g)*b'^2*l2*m3^2
-(cos(a-g))^2*sin(a-b)*a'^2*l1*m2*m3
+cos(a-b)*cos(a-g)*sin(b-g)*b'^2*l1*m3^2
+cos(a-b)*cos(a-g)*sin(a-g)*a'^2*l1*m3^2-sin(b)*m2^2*grav
-2*sin(b)*m2*m3*grav-sin(b)*m1*m3*grav-sin(b)*m1*m2*grav
+cos(b-g)*sin(g)*m3^2*grav+cos(a-b)*sin(a)*m3^2*grav
+cos(a-b)*sin(a)*m2^2*grav+(cos(a-g))^2*sin(b)*m3^2*grav
+cos(b-g)*sin(g)*m1*m3*grav+2*cos(a-b)*sin(a)*m2*m3*grav
+cos(a-b)*sin(a)*m1*m3*grav+cos(a-b)*sin(a)*m1*m2*grav
-cos(a-g)*sin(a)*cos(b-g)*m3^2*grav
+(cos(a-g))^2*sin(b)*m2*m3*grav
-cos(a-b)*cos(a-g)*sin(g)*m3^2*grav-sin(b)*m3^2*grav
+cos(b-g)*sin(g)*m2*m3*grav
-cos(a-g)*sin(a)*cos(b-g)*m2*m3*grav
```

```
-cos(a-g)*sin(a)*cos(b-g)*m1*m3*grav
-cos(a-b)*cos(a-g)*sin(g)*m2*m3*grav)

Denominator:
((m3*cos(a-g))^2*m2*l2
+m3^2*(cos(a-g))^2*l2-m1*m2*l2-m1*m3*l2+m1*m3*(cos(b-g))^2*l1
-m2^2*l2-2*m2*m3*l2+m2*m3*(cos(b-g))^2*l1-m3^2*l2
+m3^2*(cos(b-g))^2*l1-m2*l2*cos(a-b)*m3*cos(b-g)*cos(a-g)
+m2^2*l2*(cos(a-b))^2+2*m2*l2*(cos(a-b))^2*m3
-m3^2*l2*cos(a-b)*cos(b-g)*cos(a-g)+m3^2*l2*(cos(a-b))^2
-m3*l1*cos(a-g)*cos(b-g)*m2*cos(a-b)
-m3^2*l1*cos(a-g)*cos(b-g)*cos(a-b))

g'':
Numerator:
(2*m3*l1^2*cos(b-g)*m2*a'^2*sin(a-b)-2*m3*l1*b'^2*sin(b-g)*m2*l2
-2*m3*l1*a'^2*sin(a-g)*m2*l2-m3^2*l1*cos(b-g)*l3*g'^2*sin(b-g)
-2*cos(a-g)*m3*l2^2*b'^2*sin(a-b)*m2
-cos(a-g)*m3*l2^2*g'^2*sin(a-g)*m2
+l1^2*cos(b-g)*sin(a-b)*a'^2*m1*m3
+l1^2*cos(b-g)*sin(a-b)*a'^2*m1*m2-l1*b'^2*sin(b-g)*m1*m2*l2
-l1*b'^2*sin(b-g)*m1*m3*l2+l1*b'^2*sin(b-g)*m2^2*l2*(cos(a-b))^2
+l1*b'^2*sin(b-g)*m3^2*l2*(cos(a-b))^2-l1*a'^2*sin(a-g)*m1*m2*l2
-l1*a'^2*sin(a-g)*m1*m3*l2+l1*a'^2*sin(a-g)*m2^2*l2*(cos(a-b))^2
+l1*a'^2*sin(a-g)*m3^2*l2*(cos(a-b))^2
+cos(a-g)*m3^2*l2*cos(a-b)*l3*g'^2*sin(b-g)
-cos(a-g)*m3^2*l2*cos(a-b)*l1*a'^2*sin(a-b)
-cos(a-g)*m2^2*l2*cos(a-b)*l1*a'^2*sin(a-b)
-l1*cos(b-g)*sin(b-g)*g'^2*l3*m2*m3
-l1*cos(b-g)*sin(b-g)*g'^2*l3*m1*m3
+l1*cos(b-g)*cos(a-b)*sin(a-g)*g'^2*l2*m3^2
+l1*cos(b-g)*cos(a-b)*sin(a-b)*b'^2*l2*m3^2
+l1*cos(b-g)*cos(a-b)*sin(a-b)*b'^2*l2*m2^2
+2*l1*b'^2*sin(b-g)*m2*l2*(cos(a-b))^2*m3
+2*l1*a'^2*sin(a-g)*m2*l2*(cos(a-b))^2*m3
+m3^2*l1^2*cos(b-g)*a'^2*sin(a-b)-m3^2*l1*b'^2*sin(b-g)*l2
-m3^2*l1*a'^2*sin(a-g)*l2-l1*b'^2*sin(b-g)*m2^2*l2
-cos(a-g)*m3^2*l2^2*g'^2*sin(a-g)
+l1^2*cos(b-g)*sin(a-b)*a'^2*m2^2
-cos(a-g)*m3^2*l2^2*b'^2*sin(a-b)
-cos(a-g)*m2^2*l2^2*b'^2*sin(a-b)-l1*a'^2*sin(a-g)*m2^2*l2
-2*cos(a-g)*m2*l2*cos(a-b)*m3*l1*a'^2*sin(a-b)
+l1*cos(b-g)*cos(a-b)*sin(a-g)*g'^2*l2*m2*m3
+cos(a-g)*m2*l2*cos(a-b)*m3*l3*g'^2*sin(b-g)
+2*l1*cos(b-g)*cos(a-b)*sin(a-b)*b'^2*l2*m2*m3
+m3^2*grav*sin(g)*l2+grav*sin(g)*m2^2*l2
-m3^2*l1*cos(b-g)*grav*sin(b)+2*m3*grav*sin(g)*m2*l2
+grav*sin(g)*m1*m2*l2+grav*sin(g)*m1*m3*l2
-grav*sin(g)*m2^2*l2*(cos(a-b))^2
-grav*sin(g)*m3^2*l2*(cos(a-b))^2
-cos(a-g)*m2^2*grav*sin(a)*l2-cos(a-g)*m3^2*grav*sin(a)*l2
-l1*cos(b-g)*sin(b)*m2^2*grav
```

```
+2*l1*cos(b-g)*cos(a-b)*sin(a)*m2*m3*grav
+l1*cos(b-g)*cos(a-b)*sin(a)*m1*m3*grav
+l1*cos(b-g)*cos(a-b)*sin(a)*m1*m2*grav
+2*cos(a-g)*m2*l2*cos(a-b)*m3*grav*sin(b)
-2*m3*l1*cos(b-g)*m2*grav*sin(b)
+cos(a-g)*m2^2*l2*cos(a-b)*grav*sin(b)
+cos(a-g)*m3^2*l2*cos(a-b)*grav*sin(b)
-cos(a-g)*m1*grav*sin(a)*m2*l2-cos(a-g)*m1*grav*sin(a)*m3*l2
-2*cos(a-g)*m2*grav*sin(a)*m3*l2-l1*cos(b-g)*sin(b)*m1*m3*grav
-l1*cos(b-g)*sin(b)*m1*m2*grav
+l1*cos(b-g)*cos(a-b)*sin(a)*m3^2*grav
+l1*cos(b-g)*cos(a-b)*sin(a)*m2^2*grav
-2*grav*sin(g)*m2*l2*(cos(a-b))^2*m3)

Denominator:
(l3*(m3*(cos(a-g))^2*m2*l2
+m3^2*(cos(a-g))^2*l2-m1*m2*l2-m1*m3*l2+m1*m3*(cos(b-g))^2*l1
-m2^2*l2-2*m2*m3*l2+m2*m3*(cos(b-g))^2*l1-m3^2*l2
+m3^2*(cos(b-g))^2*l1-m2*l2*cos(a-b)*m3*cos(b-g)*cos(a-g)
+m2^2*l2*(cos(a-b))^2+2*m2*l2*(cos(a-b))^2*m3
-m3^2*l2*cos(a-b)*cos(b-g)*cos(a-g)+m3^2*l2*(cos(a-b))^2
-m3*l1*cos(a-g)*cos(b-g)*m2*cos(a-b)
-m3^2*l1*cos(a-g)*cos(b-g)*cos(a-b)))
```

APPENDIX E EXTERNAL BEZIER ALGORITHM IN MATLAB

Source:

http://www.mathworks.com/matlabcentral/fileexchange/26082-connecting-a-set-of-points-with-smoothly-connected-cubic-bezier-segments

Function displayBezier.m:

```
%Code subject: Simple display of a set of 2d cubic Bezier curve segments
%Programmer: Aaron Wetzler, aaronwetzler@gmail.com
%Date:12/12/2009
%
%displayBezier(knots,cp1,cp2)
%knots - Set of points with each row holding a point . In this case they must be 2-d. The
code can
%easily be changed to display 3d points as well.
%[cp1,cp2]- the set of control points used to determine the characteristics
%of the Bezier segments
%
%Each segment will be drawn with 11 subsegments i.e. straight lines

function displayBezier(knots,cp1,cp2)

n=size(cp1,1);
dim=size(cp1,2);

t=[0:0.1:1]';
t=repmat(t,1,dim);
lt=size(t,1);

cpnts=cat(3, knots(1:end-1,:), cp1, cp2, knots(2:end,:));
% hold on;
for i=1:n
    B=repmat(cpnts(i,:,1),lt,1).*((1-t).^3)  +  3*repmat(cpnts(i,:,2),lt,1).*(t.*(1-t).^2)
+ 3*repmat(cpnts(i,:,3),lt,1).*((t.^2).*(1-t))+repmat(cpnts(i,:,4),lt,1).*(t.^3);
%     plot(B(:,1),B(:,2),'r');
%     plot(knots(:,1),knots(:,2),'.b');
end
```

Function findControlPoints.m:

```
%Code subject: Finding Bezier segment control points based on smoothness
%                                   assertion for a set of arbitrary  n-d points
%Programmer: Aaron Wetzler, aaronwetzler@gmail.com
%Date:12/12/2009

%This work is entirely derived from the work of Oleg V Polikarpotchkin
%I have simply ported it from his .NET code on codeProject and applied Matlabs
%matrix features to enable it to be multi-dimensional. A known bug in the
%original codeproject.com  code  was resolved by Peter Lee.
%The original post along with the messages can be found here:
%http://www.codeproject.com/KB/graphics/BezierSpline.aspx?msg=3301993#xx3301993xx
%There is very little error checking in the code I provide so if you wish
%to reuse it make sure to add error checking.
%WARNING- The assumptions used do not guarantee relative local smoothness. There can be
loops or nearly jagged edges. If the points are well selected however these artifacts are
highly unlikely.

%[P1,  P2]=gcp(P0)
%The function expects an [n X  m] array as input where each of the n
%rows represents a point and each of its m columns are the components of
%the point. The matrix P0 to a large extent represents the control points
%on a set of cubic order Bezier segments.
%The Bezier cubic is:
%B(t)=(1-t)^3*P0+3(1-t)^2*t*P1+3(1-t)*t^2*P2+t^3*P3
%
%P1 and P2 will be the derived control points for each segment
%
%The following conditions are used to derive P1 and P2:
%1 - P1(i+1)+P2(i)=2P0(i+1)
%2 - P1(i)+2P1(i+1)= P2(i+1)+2P2(i)
%3 - 2P1(1)-P2(1)=P0(1)
%4 - 2P2(n)-P1(n)=P3(n)=P0(n+1)
%
%These are derived from the the initial assertions that
%B(i,1)`=B(i+1,0)`
%B(i,1)``=B(i+1,0)``
%B(1,0)``=B(n,1)``=0;
%
%In order to find P1 and P2 we first eliminate P2 from the conditions that
```

```
%we derive from the problem. We then can place the unknown P1 coefficients
%in a matrix with a calculable solution vector comprised of combinations of
%pairs of the input set of points P0. When we do so we see that we receive
%a tridiagonal matrix symetrical about the main diagonal.
%Finally we solve the system of equations using an algorithm for solving
%tridiagonal matrices.

function  [P1,  P2]=findControlPoints(P0)

numSegments = size(P0,1)-1;%Get the number of points and let numSegments b the number of
segments i.e. number of points- 1
dim=size(P0,2);%Get  the  number  of  dimensions  being  used.  Can  be  n-dimensional.  Only
really need up to 3 dimensions

%We want to find P1 and P2 and we start by making them all zeros
P1 = zeros(numSegments,dim);
P2 = zeros(numSegments,dim);

%Simple error check
if (numSegments < 1)
    disp('Input vector must contain at least 2 points'); return
end

 %Special case: Bezier curve should be a straight line.
if (numSegments == 1)
    P1(1,:) = (2.0 * P0(1,:) + P0(2,:)) / 3.0;  %3P1 = 2P0 + P3
    P2(1,:) = 2.0 *P1(1,:) - P0(1,:);               % P2 = 2P1 - P0
    return
end

%Set solution values
solutionVector=zeros(numSegments,dim);
solutionVector(2:numSegments-1,:)=4.0*P0(2:numSegments-1,:)+2.0*P0(3:numSegments,:);

%Set start and end values of solution vector
solutionVector(1,:) = P0(1,:) + 2 * P0(2,:);
solutionVector(numSegments,:)= (8.0 * P0(numSegments,:) + P0(numSegments+1,:)) / 2.0;

% Solve for P1
P1=solveForP1(solutionVector);
```

```
%Solve for P2
 P2(1:numSegments-1,:)=2*P0(2:numSegments,:)-P1(2:numSegments,:);
 P2(numSegments,:)=(P0 (numSegments+1,:) + P1(numSegments,:)) /2.0 ;

 %Return with P1 and P2 having been determined
return

%This function solves the known tridiagonal matrix with the precalculated
%solution vector for the problem at hand.
function P1=solveForP1(solutionVector)

n = size(solutionVector,1); %find the number of equations
dim=size(solutionVector,2); %and the number of dimensions were working in
P1 = zeros(n,dim); %initialize the result
tmp = P1;%give us a temp variable of the same size

b =ones(1,dim)* 2.0;%The algorithm is initialized with P1(1) coefficient
P1(1,:)= solutionVector(1,:) ./ b;

%no we work our way through our tridiagonal matrix
for i = 2:n
    tmp(i,:)= 1 ./ b;
    if (i<n)
        b(:)=4.0-tmp(i,:);
    else
        b(:)=3.5-tmp(i,:);
    end

    P1(i,:) = (solutionVector(i,:) - P1(i - 1,:))./ b;
end

%Here we work our way back resubstituting the intermediate solutions
for i = 2:n
    P1(n - i+1,:) =P1(n-i+1,:)- tmp(n - i+2,:) .* P1(n - i+2,:); % Backsubstitution.
end

%Return having found the solved matrix output
return
```

REFERENCES

Acary, V. (2009) “Toward higher order event–capturing schemes and adaptive time-step strategies for nonsmooth multibody systems”, INRIA Institut National De Recherche En Informatique Et En Automtique, Rapport de Recherche No 7151, December 14, 2009, ISSN 0249-6399, ISRN INRIA/RR--7151--FR+ENG

Acosta, G., Dura’n, R. G., Rossi, J. D. (2002) “An Adaptive Time Step Procedure for a Parabolic Problem with Blow-up”, Computing 68, 343–373, Digital Object Identifier (DOI) 10.1007/s00607-002-1449-x

Anitescu, M., Potra, F. A. (2002) “A time stepping method for stiff multibody dynamics with contact and friction”, Int. J. Numer. Meth. Engng 2002; 55:753–784 (DOI: 10.1002/nme.512)

Ascher, U. M.; Petzold, L. R. (1998) “Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations”, SIAM, Philadelphia, ISBN 0-89871-412-5

Bergan, P. G., Mollestad, E. (1985) “An automatic time stepping algorithm for dynamic problems”, Computer Methods in Applied Mechanics and Engineering, 49 (1985), pp. 299-318

Brenan, K. E., Campbell, S. L., Petzold, L. R. (1987) “Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations”, Society for Industrial Mathematics; 2 edition 1987, ISBN-10: 0898713536, ISBN-13: 978-0898713534

Burden, R., Faires, J. D. (2001): “Numerical Analysis”, 7th (Seventh) Edition, ASIN: B0012KS450 2001

Butcher, B. C. (1986) “Optimal order and stepsize sequences”, IMA J. Numer. Anal., 6 (1986), pp. 433-438

Califano, C., Monaco, S., Normand-Cyrot, D. (2010) “On the observer design through output scaling in discrete time”, 2010 American Control Conference, June 30 - July 02, 2010, pp. 5290 - 5295

Calvo, M., Laburta, P., Montijano, J. I., Randez, L. (2008) “On the Long Time Error of First Integrals of Some Numerical Integrators”, Numerical Analysis and Applied Mathematics, International Conference 2008

Cameron, F., Pich, R., Forsman, K. (1998) “Variable Step Size Time Integration Methods for Transient Eddy Current Problems”, IEEE TRANSACTIONS ON MAGNETICS, VOL 34, NO 5, SEPTEMBER 1998

Cardenal, J., Cuadrado, J., Morer, P., Bayo, E. (1999) “A multi-index variable time step method for the dynamic simulation of multibody systems”, International Journal For Numercial Methods In Engineering, Int. J. Numer. Meth. Engng. 44, 1579—1598 (1999)

Cash, J. R. (1976) “Semi-Implicit Runge-Kutta Procedures with Error Estimates for the Numerical Integration of Stiff Systems of Ordinary Differential Equations”, Journal of the Association for Computing Muchmery, Vol 23, No 3, July 1976, pp 455-460

Cellier, F. E. (1991), Continuous System Modeling, Springer-Verlag New York, ISBN 0-387-97502-0

Chen, Y.-W., Liu, C.-S., Chang, J.-R. (2007) “A chaos detectable and time step-size adaptive numerical scheme for nonlinear dynamical systems”, Journal of Sound and Vibration 299 (2007) 977–989

Choif, C.-K., Chung, H.-J. (1996) “ERROR ESTIMATES AND ADAPTIVE TIME STEPPING FOR VARIOUS DIRECT TIME INTEGRATION METHODS”, Computers & Structures Vol. 60. No 6. pp 923-944

Combescure, A., Gravouil, A. (2001) “A multi-time-step explicit-implicit method for non-linear structural dynamics” International Journal Numerical Methods in Engineering, Vol. 50, 199–225, 2001

Cuadrado, J., Morer, P., Bayo, e. (1999) “A MULTI-INDEX VARIABLE TIME STEP METHOD FOR THE DYNAMIC SIMULATION OF MULTIBODY SYSTEMS “,Int. J. Numer. Meth. Engng. 44, 1579-1598

De Kok, J. L., Wind, H. G. (2002) “Selecting the appropriate time step in an integrated systems model”, available from the International Environmental Modelling &

Software Society (IEMSS),
http://www.iemss.org/iemss2002/proceedings/pdf/volume%20uno/15_dekok.pdf

Doerry, N., Robey, H., Amy, J., and Petry, C. (1996) "Powering the Future with the Integrated Power System," Naval Engineering Journal, pp. 267–282, May 1996.

Dunnington, L., Stevens, H., and Grater, G. (2003) "Integrated Engineering Plant for Future Naval Combatants - Technology Assessment and Demonstration Roadmap," Tech. Rep. MSD-50-TR-2003/01, Anteon Corp., January 2003.

Emel'yanenko, V. V. (2007): "A method of symplectic integrations with adaptive time-steps for individual Hamiltonians in the planetary N-body problem", Celestial Mech Dyn Astr (2007) 98:191–202, DOI 10.1007/s10569-007-9077-6

Enright, W. H. (1974) "Second Derivative Multistep Methods for Stiff Ordinary Differential Equations", SIAM Journal on Numerical Analysis, Vol. 11, No. 2 (Apr., 1974), pp. 321-331

Fehlberg, E. (1969) "Low-order classical Runge-Kutta formulas with step size control and their application to some heat transfer problems", NASA Technical Report 315.

Fehlberg, E. (1970) "Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme", Computing (Arch. Elektron. Rechnen), vol. 6, pp. 61-71.

Fujimoto, R. M. (2000) "Parallel and Distributed Simulation Systems". New York, NY: Wiley

Gear, W. C. (1971) "Numerical Initial Value Problems in Ordinary Differential Equations", Prentice Hall 1971, ISBN-10: 0136266061, ISBN-13: 978-0136266068

Gilmore, M. (2005) "The Navy's DD(X) Destroyer Program." Congressional Budget Office testimony, July 2005.

Goeken, D., Johnson, O. (2000) "Runge–Kutta with higher order derivative approximations", Applied Numerical Mathematics 34 (2000) 207–218

González-Parra, G., Arenas, A. J., Chen-Charpentier, B. M. (2010) “Combination of nonstandard schemes and Richardson’s extrapolation to improve the numerical solution of population models”, Mathematical and Computer Modelling 52 (2010) 1030–1036

Gordon, B. W., and Asada, H. H., (2000) ‘‘Modeling, Realization, and Simulation of Thermo-Fluid Systems Using Singularly Perturbed Sliding Manifolds,’’ ASME J. Dyn. Syst., Meas., Control, 122, pp. 699–707

Greenspan, D., Hafner, W., Rebaric, M. (1965) “On varying stepsize in numerical integration of first order differential equations”, Numer. Math., 7 (1965), pp. 286-291

Gu, B., Asada, S. (2004) “Co-Simulation of Algebraically Coupled Dynamic Subsystems Without Disclosure of Proprietary Subsystem Models”, Journal of Dynamic Systems, Measurement, and Control, March 2004, Vol. 126 / 1

Günther, M., Kværnø, A., Rentrop, P. (2001) “Multirate partitioned Runge-Kutta methods”, BIT, 41, 504–514 (2001)

Gupta, G. K., Wallace, C. S. (1979) “A New Step-Size Changing Technique For Multistep Methods”, Mathematics of Computation, Vol. 33, Nr. 145, Jan. 1979, pp. 125-138

Hairer, E., Norsett, S. P., Wanner, G. (1987) “Solving Ordinary Differential Equations, I. Nonstiff Problems”. Springer Series in Comp. Math., Vol. 8, Springer Verlag, Berlin–Heidelberg–New York–London–Tokyo

Hairer, E., Wanner, G. (1996) “Solving Ordinary Differential equations II - Stiff and DAE problems”, Springer; 2nd edition (September 20, 1996), ISBN-10: 3540604529, ISBN-13: 978-3540604525

Hensen J.L.M. (1999) “A comparison of coupled and de-coupled solutions for temperature and air flow in a building”, in ASHRAE Transactions vol. 105:2, pp. 962-969

Hussain, M. Z., Hussain, M. (2009) “Shape Preserving Scattered Data Interpolation”, European Journal of Scientific Research, ISSN 1450-216X Vol.25 No.1 (2009), pp.151-164

Hutchinson, T. A. (2009) “An Adaptive Time-Step for Increased Model Efficiency”, Weather Services International, Andover, Massachusetts

Jansson, J., Logg, A. (2004) “Simulation of Mechanical systems with Individual Time Steps”, Simula Research Laboratory publication, http://home.simula.no/~logg/pub/thesis/compact-disc/papers/pdf/paper-10.pdf

Jansson, J., Logg, A. (2008) “Algorithms and Data Structures for Multi-Adaptive Time-Stepping”, ACM transactions on mathematical software [0098-3500], 2008 vol:35 iss:3 pg:1

Johnson, C. (1988) “Error Estimates and Adaptive Time-Step Control for a Class of One-Step Methods for Stiff Ordinary Differential Equations”, SIAM Journal on Numerical Analysis, Vol. 25, No. 4 (Aug., 1988), pp. 908-926

Joukhadar , A., Laugier, C. (1996) “Adaptive time step for fast converging dynamic simulation system”, Proc. IROS 96 0-7803-3213-X/96/, IEEE 1996, pp. 418-424

Karafyllis, I., Kravaris, C. (2008) “From continuous-time design to sampled-data design of nonlinear observers”, 2008 47th IEEE Conference on Decision and Control [1-4244-3123-9], pp. 5408

Kofman, E., Junco, S. (2001) “Quantized State Systems: A DEVS Approach for Continuous System Simulation”, Transactions of the Society for Computer Simulation International, Volume 18, Issue 3, pp. 123 – 132, ISSN:0740-6797

Krogh, F. T. (1973) “Algorithms for Changing the Step Size”, SIAM Journal on Numerical Analysis, Vol. 10, No. 5 (Oct., 1973), pp. 949-965

Kubler, R., Schiehlen, W. (2000) “Modular Simulation in Multibody System Dynamics”, Multibody System Dynamics 4: 107–127

Kulikov, G. Yu., Khrustaleva, E. Yu. “Automatic Step Size and Order Control in Implicit One Step Extrapolation Methods” Zh. Vychisl. Mat. Mat. Fiz. 48, 1580–1606 (2008) [Comput. Math. Math. Phys. 48, 1545 - 1569 (2008)].

Kulikov, G. Yu., Khrustaleva, E. Yu. (2010) “Automatic Step Size and Order Control in One Step Collocation Methods with Higher Derivatives”, ISSN 0965-5425, Computational Mathematics and Mathematical Physics, 2010, Vol. 50, No. 6, pp.

1006 - 1023. (C) Pleiades Publishing, Ltd., 2010. Original Russian Text © G.Yu. Kulikov, E.Yu. Khrustaleva, 2010, published in Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 2010, Vol. 50, No. 6, pp. 1060–1077

Law, A. M., Kelton, W. D. (2000) "Simulation Modeling and Analysis", 3rd ed., New York, NY: McGraw-Hill

Lee, S., Pritchett, A., Goldsman, D. (2001) "Hybrid Agent-Based Simulation For Analyzing The National Airspace System", Proceedings of the 2001 Winter Simulation Conference

Lether, F. G. (1965) "The Use of Richardson Extrapolation in One-Step Methods with Variable Step Size", Mathematics of Computation, Vol. 20, No. 95, Jul., 1966, pp. 279-285

Lew, A., Marsden, J. E., Ortiz, M., West, M. (2003) "Asynchronous variational integrators", Archive for Rational Mechanics and Analysis, 167(2):85–146, 2003

Lew, A., Marsden, J. E., Ortiz, M., West, M. (2004) "Variational time integrators", International Journal for Numerical Methods in Engineering, 60(1):153–212, 2004

Lindberg, B. (1977) "Characterization of optimal stepsize sequences for methods for stiff differential equations", SIAM J. Numer Anal., 14 (1977), pp. 858-887

Lively, K. A., Scheidt, D. H., and Drew, K. F. (2005) "Mission Based Engineering Plant Control,", ASNE Rconfiguration and Survivability Symposium, February 2005.

Logg, A. (2004) "Multi-adaptive time integration", Applied Numerical Mathematics 48 (2004) 339–354

Lopes, G., Bermudez, M. (2001) "Evaluation and design of variable step size adaptive algorithms", IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221) [0-7803-7041-4] Lopes yr:2001 pg:3845

Lowan, A. N. (1960) "On the Propagation of Round-Off Errors in the Numerical Integration of the Heat Equation", Mathematics of Computation, Vol. 14, No. 70 (Apr., 1960), pp. 139-146

Mahjoubi, N., Gravouil, A., Combescure, A., Greffet, N. (2008) "A generalized multi-time-step method for a wide range of numerical scheme applied to transient nonlinear structural dynamics", 8th. World Congress on Computational Mechanics (WCCM8), June 30 - July 5, 2008, Venice, Italy

Meurer, T., Graichen, K., Gilles, E.-D. (2005) "Control and Observer Design for Nonlinear Finite and Infinite Dimensional Systems", Springer, ISBN-10: 9783540279389, ISBN-13: 978-3540279389, ASIN: 3540279385

Mielke, A. (2006) "Analysis, modeling and simulation of multiscale problems", Berlin, Springer, 2006 QA401.A52x 2006

Mihai, L. A., Ainsworth, M. (2009) "An adaptive multi-scale computational modelling of Clare College Bridge", Comput. Methods Appl. Mech. Engrg. 198 (2009) 1839–1847

Morrison, D. (1962) "Optimal mesh size in the numerical integration of an ordinary differential equation", J. Assoc. Comput. Mech., 9 (1962), pp. 98-103

Nairouz, B., Hoepfer, M. (2009): "Investigations for Time Step Settings in a Dynamic System Co-Simulation Environment", Electric Ship Design Symposium, Washington D.C., Jan 2009

Nakano, A., Vashishta, P., Kalia, R. K. (1993) "Parallel multiple-time-step molecular dynamics with three-body interaction", Computer Physics Communications 77 (1993), pp. 303-312

Nakashima, M. (1972) "On the Propagation o f Error in Numerical Integration", Proc. Japan Acad., 48 (1972) [Vol. 48, No. 7], pp. 484-488

Ortega, J. M., Rheinboldt, W. C. (1987) "Iterative Solution of Nonlinear Equations in Several Variables", Society for Industrial Mathematics 1987, ISBN-10: 0898714613, ISBN-13: 978-0898714616

Pfau, R. U. (2007) "A priori step size adaptation for the simulation of non-smooth systems", Commun. Numer. Meth. Engng 2007; 23:85–96

Pham, T., Oudin-Dardun, F. (2009) “C(p, q, j) Scheme with Adaptive time step and Asynchronous Communications”, Parallel Computational Fluid Dynamics 2007, DOI: 10.1007/978-3-540-92744-0_41, pp. 329-337

Piah, A. R. Mt., Saaban, A., Majid, A. Abd. (2006) “Range restricted positivity-preserving scattered data interpolation”, Journal of Fundamental Sciences, Vol 2, No 1-2

Pritchett, A. R., Lee, S. M., Goldsman, D. (2001) “Hybrid-system simulation for safety analysis of the National Airspace System” Accepted for publication in AIAA Journal of Aircraft

Richardson, L. F. (1911). "The approximate arithmetical solution by finite differences of physical problems including differential equations, with an application to the stresses in a masonry dam". Philosophical Transactions of the Royal Society of London, Series A 210 (459-470): 307–357. doi:10.1098/rsta.1911.0009.

Rios Neto, A., Rama Rao, K. (1990) “A stochastic approach to global error estimation in ODE multistep numerical integration”, Journal of Computational and Applied Mathematics 30 (1990) 257-287

Ruge, P. (1999) “A priori local error estimation with adaptive time-stepping”, Communications In Numerical Methods In Engineering, Commun. Numer. Meth. Engng. 15, pp. 479-491

Saha, P., Tremaine, S. (1994) “Long-term planetary integration with individual time steps”, The Astronomical Journal, 1994

Savcenco , V., Mattheji, R. M. M. (2010) “Multirate numerical integration for stiff ODEs”, Progress in Industrial Mathematics at ECMI 2008 [3-642-12110-1] 2010 pg:327

Schröder, D., Lenz, U., Beuschel, M., Hangl, F. D., Frenz, T., Strobl, D., Straub, S., Fischle, K., Rau, M., Angermann, A. (2010) “Intelligent Observer and Control Design for Nonlinear Systems”, Springer, ISBN-10: 3642083463, ISBN-13: 978-3642083464

Shin, M., West, M. (2008) “High-order Multistep Asynchronous Splitting Integrators (MASI)”, 8th. World Congress on Computational Mechanics (WCCM8), 5th

European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2008), June 30 - July 5, 2008, Venice, Italy

Sigal, G. (2005) "On High Order Strong Stability Preserving Runge–Kutta and Multi-Step Time Discretizations ", Journal of Scientific Computing, Vol. 25, Nos. 1/2, November 2005

Stoer, J., Bulirsch, R. (2002) "Introduction to Numerical Analysis", Springer; 3 edition 2002, # ISBN-10: 038795452X, # ISBN-13: 978-0387954523

Struler, E. de, Hoefliger, J., Kale, L.V., Bhandarkar, M. (2000) "A new approach to software integration frameworks for multiphysics simulation codes", Proceedings of IFIP, TC2/WG2.5, Working Conference on Architecture of Scientific Software, pp. 87-104, Ottawa, Canada.

Süli, E., Mayers, D. F. (2003) "An Introduction to Numerical Analysis", Cambridge University Press (September 8, 2003), ISBN-10: 0521007941, ISBN-13: 978-0521007948

Tomulik, P., Fraczek, J. (2010) "Simulation of multibody systems with the use of coupling techniques: a case study", Multibody Syst Dyn., vol:25 iss:2 pg:145, DOI 10.1007/s11044-010-9206-y

Trckal, M., Wetter, M., Hensen, J. (2007) "Comparison of Co-Simulation Approaches for Building and HVAC/R system Simulation", Proceedings: Building Simulation 2007

Tseng, C.-C., Lee, S.-L. (2008) "Design of digital differentiator using difference formula and Richardson extrapolation", IET Signal Process., 2008, Vol. 2, No. 2, pp. 177-188, doi:10.1049/iet-spr:20070145

Utumi, M., Takaki, R., Kawai, T. (1996) "Optimal Time Step Control for the Numerical Solution of Ordinary Differential Equations", SIAM Journal on Numerical Analysis, Vol. 33, No. 4 (Aug., 1996), pp. 1644-1653

Volovoi, V. (2004), "Modeling of System Reliability Using Petri Nets with Aging Tokens", Reliability Engineering and System Safety, Vol. 84, 2004, pp. 149-161

Volovoi, V. (2006), “Stochastic Petri Nets Modeling using SPN@,” Proceedings of RAMS-2006 Symposium, Newport Beach, CA, January 26–29, 2006; Paper 2006RM-166

Walks, J. P. and Mearman, J. F. (2005) “Integrated Engineering Plant,” ASNE Reconfiguration and Survivability Symposium, February 2005.

Wang, H., Taylor, S., Simkin, J., Biddlecombe, C., Trowbridge, B. (2001) “An Adaptive-Step Time Integration Method Applied to Transient Magnetic Field Problems”, IEEE TRANSACTIONS ON MAGNETICS, VOL. 37, NO. 5, SEPTEMBER 2001

Wang, J., Ma, Z. D., and Hulbert, G. M., (2003) ‘‘A Gluing Algorithm for Distributed Simulation of Multibody Systems,’’ Nonlinear Dyn., 34(1-2), pp. 159–188.

Wang, J., Ma, Z.D., Hulbert, G. (2005) “A distributed mechanical system simulation platform based on a “gluing algorithm””. J. Comput. Inf. Sci. Eng. 5, 71–76 (2005)

Wendland, H. (2005) “Scattered Data Approximation”, Cambridge University Press; ISBN-10: 0521131014, ISBN-13: 978-0521131018

Werschul, A. G. (1979) “Optimal Order for Approximation of Derivatives”, JOURNAL OF COMPUTER AND SYSTEM SCIENCES 18, 213-217 (1979)

Wirth, N. (1995) "A Plea for Lean Software," Computer, vol. 28, no. 2, pp. 64-68, Feb. 1995, doi:10.1109/2.348001)

Yoo, S., Choi, K. (2000) “Optimizing Timed Cosimulation by Hybrid Synchronization“, Design Automation for Embedded Systems, 5, 129–152 (2000), Kluwer Academic Publishers, Boston

Zadunaisky, P. E. (1976) “On the Estimation of Errors Propagated in the Numerical Integration of Ordinary Differential Equations”, Numer. Math. 27, 21-39

Zeigler, B., Kim, T. G., Praehofer, H. (2000) ”Theory of Modeling and Simulation – Second Edition”, New York, Academic Press, 2000

Zeigler, B., Lee, J. S. (1998): ”Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment”, SPIE Proceedings Vol. 3369, pp. 49-58, 1998

Zhai, Z. (2004) “Developing an Integrated Building Design Tool by Coupling Building Energy Simulation and Computational Fluid Dynamics Program”, PhD thesis, Massachusetts Institute of Technology.

Zhang, H., Cui, P., Wang, H. (2010) “An Effective Interaction Control Mechanism with Minor Step for Multidisciplinary Collaborative Simulation System”, Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design, 978-1-4244-6763-1/10 2010 IEEE